Escuela Politécnica Superior

Plataforma de gestión y visualización de dispositivos IoT

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Ángela Serna Chávez

Tutor/es:

José Ángel Berna Galiano

Julio 2020

Universitat d'Alacant
Universidad de Alicante

Índice

1 Justificación y objetivos

La Transformación Digital de las actividades económicas es un objetivo fundamental de las sociedades actuales para conseguir mejoras en la productividad, sostenibilidad con el medio ambiente y generar nuevas fuentes de riqueza. El paradigma IoT (Internet of Things) es un elemento fundamental en la transformación de los sistemas productivos, donde el despliegue de sensores permite la captación de gran cantidad de datos del mundo físico.

Este proyecto nace como idea a partir de otro proyecto de investigación previo[1] sobre la conectividad y comunicación para dispositivos IoT. En esta investigación se presenta un sistema formado por una estación base que recibe paquetes de información desde dispositivos IoT (sensores) a través de un nuevo sistema de comunicación Wi-Fi patentado que proporciona un consumo bajo de recursos en los sensores, de coste nulo y más seguro.

A partir de este trabajo de investigación se plantea la posibilidad de extender el sistema propuesto para ser capaces de visualizar y acceder a la información enviada por los sensores a las estaciones base con ese novedoso sistema de comunicación desde cualquier lugar con conectividad a Internet.

El TFG plantea el desarrollo de una plataforma de gestión y visualización de información de diferentes proyectos IoT. Cada proyecto IoT consta de un despliegue de sensores en un entorno físico que envían sus datos a estaciones base utilizando el sistema de comunicación antes mencionado.

La plataforma definirá los protocolos de comunicación entre las estaciones base y el sistema de gestión de datos. Así mismo, la plataforma definirá un sistema de visualización de los datos de cada proyecto empleando tecnología Web, permitiendo su acceso multiplataforma (PC, smartphone, tablet) desde cualquier emplazamiento con conectividad a Internet.

[1] https://rua.ua.es/dspace/handle/10045/101573

2 Agradecimientos

Me gustaría agradecer a mi tutor, José Ángel Berna Galiano, por todo el apoyo y ánimos mostrado desde el principio, además de su gran disposición en cualquier momento para ver mis propuestas y aconsejarme siempre lo mejor y ayudarme en cualquier cosa que he necesitado.

También me gustaría agradecer el apoyo y consejos brindados a lo largo del proyecto a Alejandro José Box Cerdá, desarrollador en SolidQ (grupo Verne). Su experiencia con las tecnologías utilizadas en el proyecto ha resultado de gran ayuda para resolver algunas de las problemáticas que han ido surgiendo a lo largo del proyecto.

3 Estado del arte

3.1 Protocolos de comunicación IoT

Los protocolos de comunicación IoT cada vez son más frecuentes y numerosos, es por ello por lo que se debe realizar un estudio previo de las diferentes opciones y ver la que mejor se adapta a las necesidades de cada proyecto. En este apartado, se comenta uno de los protocolos más utilizados hoy en día y el protocolo que se ha utilizado para implementar la solución.

3.1.1 MQTT (Message Queue Telemetry Transport)

MQTT [1] [2] es un protocolo de comunicación ligero M2M (Machine To Machine) basado en el patrón publicación-suscripción sobre TCP/IP inventado por Andy Stanford-Clark de IBM y Arlen Nipper en 1999; actualmente la versión 5.0[2] del protocolo es un estándar OASIS[3].

En MQTT se distinguen dos entidades: un servidor que hace de intermediario (conocido como *Broker*) y se encarga de recibir y redirigir los mensajes de los clientes origen a los de destino. Se considera cliente a cualquier cosa que interactúe con dicho servidor para enviar y/o recibir mensajes.

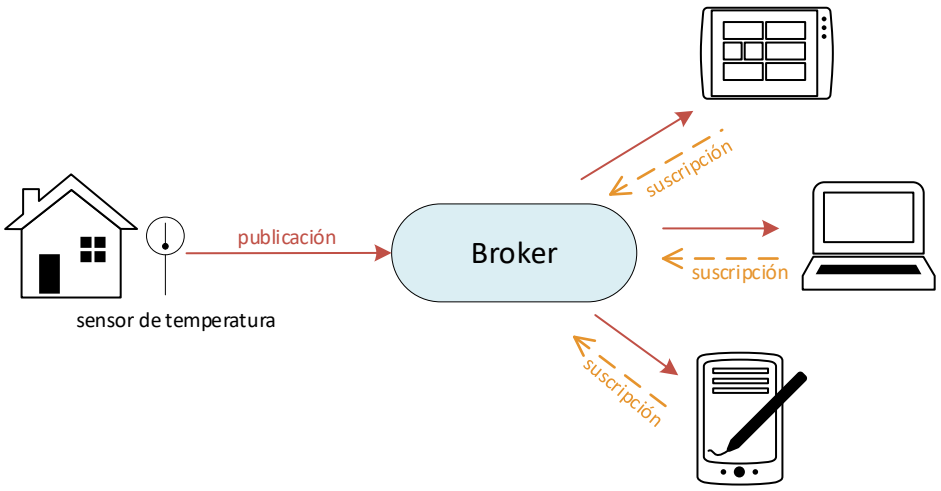


Figura 1. Ejemplo de funcionamiento de MQTT

El funcionamiento básico general de MQTT, como se muestra en la Figura 1, consiste en uno o varios clientes publican información relacionada con un tema concreto en el *Broker* y este lo distribuye a aquellos clientes suscritos a ese tema.

[2] https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html
[3] https://www.oasis-open.org/

En la Figura 1 las líneas de color naranja oscuro representan los datos publicados por el sensor y las de color naranja claro equivalen a la suscripción de los diferentes dispositivos o clientes.

3.1.2 CoAP (Constrained Application Protocol)

CoAP [3][4] [5] es un protocolo de comunicación máquina a máquina (M2M) especialmente diseñado para transferencia web en dispositivos con recursos hardware (ROM, RAM …) y de red limitados (sensores de baja potencia, por ejemplo), diseñado por el IETF[4] y especificado en la RFC 7252 [6].

CoAP se basa en UPD y funciona sobre dispositivos que soportan este protocolo. Además, utiliza el modelo RESTful web de HTTP con cabeceras reducidas, tamaños de mensajes pequeño.

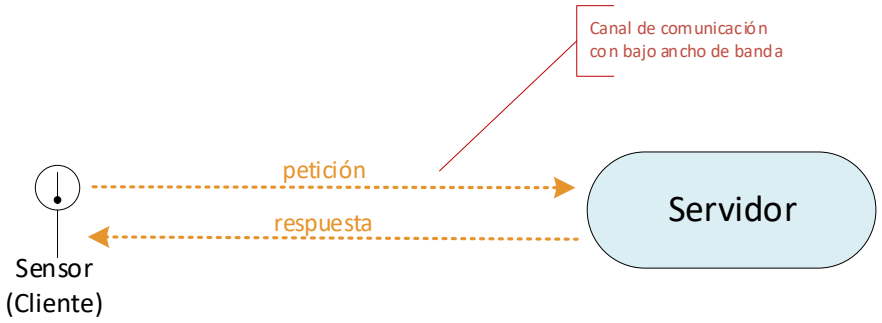


Figura 2. Ejemplo de funcionamiento de CoAP

El funcionamiento de CoAP es, básicamente, el de una arquitectura cliente-servidor (Figura 2) que soporta las operaciones GET, PUT, POST y DELETE. El cliente realiza una petición y el servidor puede responder con determinados códigos definidos en la RFC 7252 [7].

A continuación, se detallan algunos de los códigos de respuesta definidos en la RFC:

2.04 Changed: se corresponde con la respuesta HTTP 204 "No Content". Esta respuesta se devuelve a peticiones de tipo POST/PUT cuando el servidor ha procesado correctamente la petición, pero no necesita devolver datos al cliente.

2.05 Content: se corresponde con la respuesta HTTP 200 "Ok", pero solo responde a peticiones de tipo GET. Se ofrece esta respuesta cuando el servidor ha procesado correctamente la petición recibida. Además, contiene una representación del recurso solicitado por la petición.

4.00 Bad Request: se corresponde con la respuesta HTTP 400 "Bad Request" y se da cada vez que el servidor no puede interpretar la petición entrante.

[4] https://www.ietf.org/standards/

4.04 Not Found: se corresponde con la respuesta HTTP 404 “Not Found”. Esta respuesta es enviada por el servidor cuando éste no es capaz de encontrar el recurso solicitado.

En el proyecto, se han utilizados los códigos 2.04 Changed y 4.00 Bad Request para indicar peticiones correctas o incorrectas respectivamente (ver apartado 5.3).

3.1.3 MQTT VS. CoAP

Ambos protocolos están orientados a la comunicación de dispositivos IoT, pero presentan diferencias que los haces más favorables para distintas soluciones [8].

MQTT está más orientado a la comunicación bidireccional entre dispositivos IoT debido a la arquitectura que presenta (publicación/suscripción); un ejemplo es una aplicación de mensajería chat. Además, está basado en TCP, por lo que en entornos con conectividad limitada puede provocar una mayor cogestión en la red.

Por otro lado, CoAP es ideal para entornos basados en servicios web (petición/respuesta HTTP) donde los clientes presentan recursos hardware o de red limitados ya que hace uso del protocolo UDP.

En el proyecto se pretende enviar datos desde una estación base (con hardware y ancho de banda limitados) a una API para almacenar toda la información en una base de datos. Debido a la situación de las estaciones base y al tipo de comunicación que se necesita (unidireccional estación base – API) el protocolo CoAP resulta más favorable. Además, presenta una implementación estándar en el *framework* de desarrollo utilizado (ver apartado 3.4.1) que facilita el desarrollo del proyecto.

3.2 Patrones de desarrollo software

3.2.1 MVVM (Model-View-ViewModel)

Model-View-ViewModel es un patrón de arquitectura software que pretende separar la lógica de negocio y de presentación de la interfaz de usuario en el desarrollo de una aplicación. Este patrón está formado por tres elementos: el modelo, la vista y el modelo de vista. La vista y el modelo están totalmente desacoplados por el modelo de la vista [9] [10] [11].

La vista se corresponde con la interfaz de usuario; define la estructura y la apariencia de lo que el usuario ve. No contiene código relacionado con la lógica de negocio, esto forma parte de las funciones del modelo de vista.

El modelo representa la capa de datos y no puede contener código que afecte a la visualización de los datos. Este elemento se usa frecuentemente junto a servicios para encapsular el acceso a los datos; de esta forma el modelo de vista puede acceder a la información a través de estos servicios.

El modelo de vista actúa como intermediario entre el modelo y la vista; se encarga de procesar las peticiones de la vista al modelo y, cuando es necesario, convierte los datos procedentes del modelo para que la vista los pueda consumir fácilmente.

En la Figura 3 se puede ver un esquema del funcionamiento básico de este patrón.

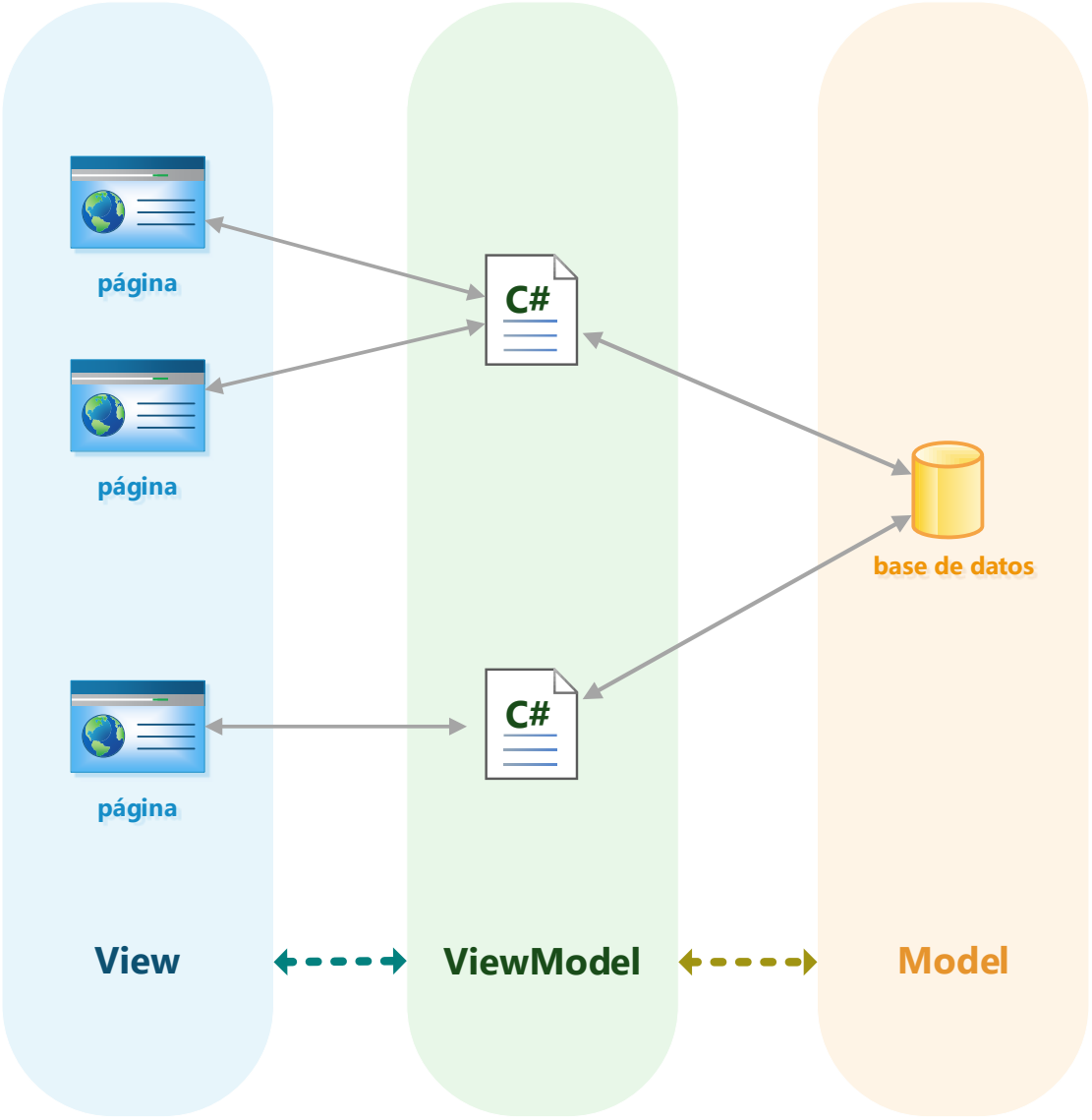


Figura 3. Patrón MVVM

3.3 Algoritmos de cifrado-descifrado

3.3.1 AES (Advanced Encryption Standard)

Se trata de un algoritmo de cifrado de clave simétrica. Este algoritmo [5]se basa en cifrar y descifrar un mensaje de cualquier tamaño con la misma clave y su vector de inicialización.

Es un algoritmo rápido y que permite el cifrado/descifrado de grandes cantidades de datos; sin embargo, sus claves son más vulnerables a ataques por fuerza bruta.

3.3.2 RSA (Rivest Shamir Adleman)

Se trata de un algoritmo de cifrado de clave asimétrica. Este algoritmo se basa en un par de claves pública/privada. La clave pública es la que se emplea para cifrar el mensaje que se va a enviar y la clave privada (más robusta y que no debe ser compartida) se emplea para descifrar dicho mensaje. El tamaño del mensaje a cifrar está condicionado por la longitud de la clave.

RSA[6] es un algoritmo muy seguro debido a la robustez de su clave privada, pero presenta limitaciones en el tamaño de mensaje a cifrar y en la cantidad de recursos necesarios para ejecutar el cifrado/descifrado (es más costoso computacionalmente).

3.3.3 AES VS RSA

La combinación de ambos métodos es muy efectiva ya que consigue obtener los beneficios de ambas partes eliminando sus puntos débiles. HTTPS (SSL) hace uso de RSA para compartir la clave simétrica que se utiliza durante el resto de la comunicación; AES es uno de los algoritmos de clave simétrica soportados por SSL. Por lo que la combinación de ambos es una solución ampliamente aceptada.

Para establecer una comunicación segura a través de CoAP se ha creado un mecanismo de cifrado personalizado que utiliza ambos algoritmos. Se comenta en profundidad en el apartado 5.3.

3.4 .NET Core

Antes de comentar .Net Core es necesario conocer .NET Framework, marco de trabajo previo a .NET Core e implementación original de .NET.

.NET Framework [12] es un marco de trabajo que fue lanzado en 2002 para el desarrollo de aplicaciones de escritorio, páginas web, etc. solamente para Windows parcialmente abierto (hay partes que pueden ser consumidas solo bajo licencia) para los lenguajes de programación C#, F# y Visual Basic. Las librerías .NET Framework aportan un gran conjunto de APIs y tipos de datos comunes que permiten construir

[5] https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf
[6] https://web.archive.org/web/20051029040347/http://rsasecurity.com/rsalabs/node.asp?id=2125

aplicaciones con diversas funcionalidades, sin la necesidad de que el desarrollador deba implementarlas; esto, a su vez, supone una gran desventaja en algunos casos ya que las librerías tienen un gran tamaño y no todas las aplicaciones van a requerir de todas las funcionalidades ofrecidas.

Como alternativa y solución mejorada a .NET Framework nace .NET Core en junio de 2016 [13].

.NET Core [14] es una plataforma de desarrollo creada por Microsoft bajo la licencia MIT o Apache 2 de código abierto[7] multiplataforma que permite crear aplicaciones para los principales sistemas operativos: Windows, Linux y macOS. Se trata de un marco de trabajo novedoso y modular: es un único bloque con funcionalidad básica extensible a través de paquetes más pequeños con una funcionalidad más específica publicados a través de NuGet. A diferencia de su predecesor, .NET Core permite compilar, ejecutar, publicar o incluso testear las aplicaciones desde la línea de comandos.

Actualmente la última versión estable es .NET Core 3.1, pero la próxima actualización va a ser directamente .NET 5.0[8] en noviembre de 2020. Con esta nueva versión Microsoft pretender seguir manteniendo los beneficios de .NET Core 3.1 (código abierto, multiplataforma, modular, ejecución y construcción desde línea de comandos…) pero yendo un paso más allá; .NET 5.0 pretende unificar el desarrollo de código independientemente del tipo de aplicación que programe y soporta un mayor número de sistemas operativos (Windows, Linux, macOs, iOs, Android…). Además, ofrece interoperabilidad con otros lenguajes como Java o Swift y Objective-C en algunos sistemas operativos.

3.4.1 CoAP en .NET Core

CoAP.NET[9] es una librería de código abierto que provee una implementación en C# de los servicios basados en CoAP para aplicaciones .NET.

Para este proyecto se ha utilizado la versión de este paquete para .NET Core: CoAP.NET.Core [15][16].

3.4.2 AES/RSA en .NET Core

.NET provee una implementación propia de algunos servicios criptográficos en su espacio de nombres "System.Security.Cryptography". Aquí se encuentran las clases de AES [17] y RSA [18] para utilizar estos algoritmos de cifrado.

[7] https://github.com/dotnet/core
[8] https://devblogs.microsoft.com/dotnet/introducing-net-5/
[9] https://github.com/smeshlink/CoAP.NET

Para los algoritmos de cifrado y descifrado utilizados en este proyecto se han utilizados las librerías nativas de Microsoft anteriormente comentadas y en ningún momento se ha recurrido a implementaciones externas, por lo que se puede asumir que esta es la implementación más efectiva y eficiente de los algoritmos.

3.4.3 Log de errores (Serilog)

Serilog [19] es una librería de código abierto[10] bajo la licencia Apache 2 para .NET que sobrescribe el elemento ILogger predeterminado para registrar la información permitiendo así a los desarrolladores personalizar el log de eventos según sus necesidades. Por ejemplo, añadir información más personalizada y detallada de cada registro y especificar si el registro se muestra por consola o se guarda en un fichero (pudiendo establecer en este caso el directorio de dicho archivo). Además, permite unificar el log de errores entre diferentes extensiones dentro de un mismo proyecto/solución.

Serilog.AspNetCore es el paquete NuGet disponible para desarrollar en .NetCore y es el utilizado en la implementación del proyecto [20] [21].

3.4.4 Dapper

Dapper [22] es un micro-ORM para C# que permite ejecutar consultas fácilmente contra cualquier tipo de base de datos. Una de las principales ventajas que presenta es que permite al desarrollador escribir las consultas SQL que se ejecutarán; por lo tanto, el rendimiento no depende de la implementación de la librería (como en Entity Framework) sino de la calidad de la consulta definida.

Otra de las ventajas más importantes de Dapper es que está protegido ante inyección de SQL ya que todas las consultas están parametrizadas, evitando así la necesidad de tener que concatenar código SQL.

Dapper también permite trabajar con transacciones. Esto permite al desarrollador aislar conjuntos de consultas para asegurar la atomicidad de las operaciones definidas en los repositorios (capa de acceso a datos).

3.5 Blazor

Blazor [23] [24] es un *framework* de código abierto[11] de Microsoft (forma parte de ASP.NET Core) para crear aplicaciones SPA (aplicaciones de página única). Las aplicaciones Blazor están formadas por componentes; un componente es un elemento de interfaz de usuario reutilizable que puede contener código C#, de marcado (HTML) e incluso otros componentes.

[10] https://github.com/serilog/serilog
[11] https://github.com/dotnet/aspnetcore/tree/master/src/Components

Este marco de trabajo se centra en ofrecer un desarrollo full-stack en C#, en contraposición a otros marcos de trabajo como Angular o React que hacen uso de JavaScript. Esto aporta ventajas como poder compartir código entre cliente y servidor o bien, reutilizar código existente en las capas de presentación.

A pesar de que el objetivo principal de Blazor es el desarrollo en C# también posibilita el uso de bibliotecas o APIS JavaScript cuando es necesario. Los componentes pueden interoperar con JavaScript en dos sentidos: desde C# se puede llamar a funciones JavaScript o bien, desde código JavaScript puede llamar a código C#.

Las aplicaciones Blazor se pueden ejecutar en el lado del cliente (*client-side*, basado en WebAssembly[12]) o en el lado del servidor (*server-side*) con el mismo propósito: implementar la lógica de presentación de la aplicación con C#. En la opción *client-side* (Figura 4. Representación de Blazor WebAssembly) la aplicación junto a sus dependencias y el entorno de ejecución en .NET se descargan en el explorador y se ejecuta [25] [26].

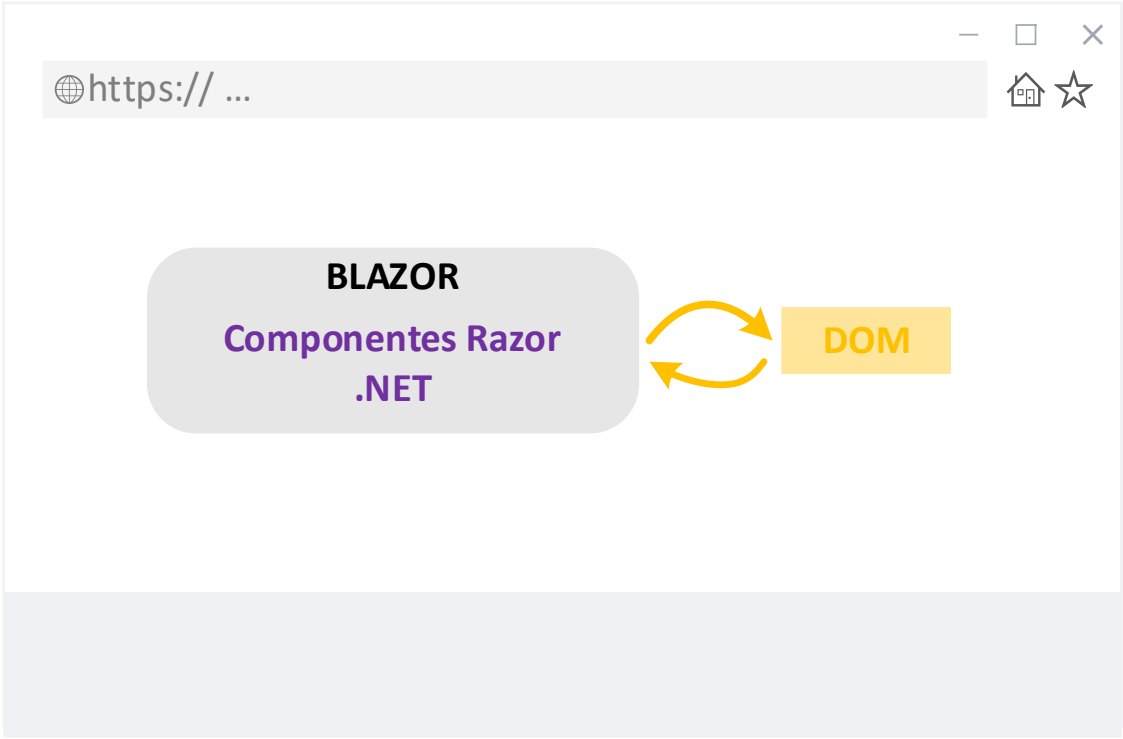


Figura 4. Representación de Blazor WebAssembly

[12] https://webassembly.org/

En el caso de la opción *server-side* (Figura 5. Representación de Blazor server-side) la aplicación se ejecuta completamente en el servidor y se comunica con el navegador a través de una conexión SignalR[13]. Cuando se ejecuta una aplicación con este modelo de hospedaje (en el servidor), primero se compila el componente solicitado en HTML en el servidor y después se envía al cliente donde se representa. Cuando se actualiza la interfaz de usuario (hacer clic en un botón, por ejemplo), el servidor realizar los cambios y solamente se envían, a través de la conexión SignalR, esos cambios sin necesidad de recargar toda la página.

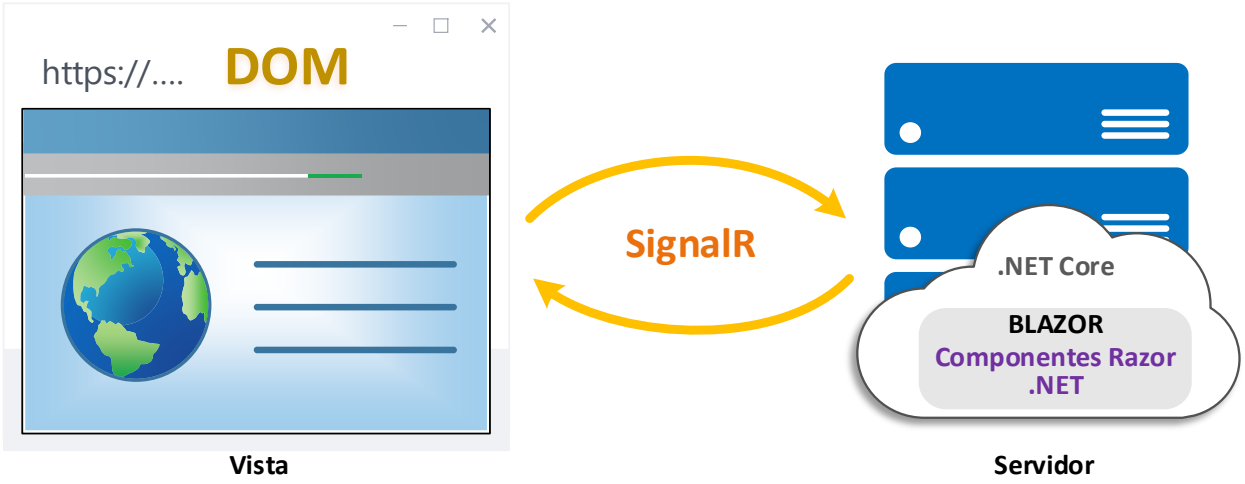


Figura 5. Representación de Blazor server-side

Independientemente de si la aplicación se ejecuta en el lado del cliente o en el del servidor, el código es el mismo, por lo que es posible pasar de una versión a la otra sin necesidad de realizar grandes cambios en el código.

La versión *server-side* fue publicada en septiembre de 2019 y es la que se ha utilizado para el desarrollo de este proyecto. La versión *client-side* se ha publicado en mayo de 2020[14]. Cuando se ejecuta la aplicación en el lado del servidor éste recibe más carga, pero el código no se envía al cliente, sino que se envía directamente la página HTML, de esta forma el cliente no puede decompilar y ver el código; en el caso de que sea necesario modificar de forma dinámica la página web (acciones usualmente ejecutadas con JavaScript en el resto de las webs) se sigue ejecutando en el lado del servidor. En el modelo *client-side* el *client-side* el servidor recibe menos carga, pero el código está comprometido al poder ser visualizado por el usuario.

[13] https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-3.1
[14] https://devblogs.microsoft.com/aspnet/blazor-webassembly-3-2-0-now-available/

3.5.1 Syncfusion

Syncfusion [27] es un proveedor de componentes y bibliotecas de desarrollo empresarial; proporciona una amplia gama de interfaces de usuario, informes e inteligencia empresarial en todas las plataformas principales de Windows y posee un equipo encargado de lanzar actualizaciones trimestrales además de dar soporte a sus productos.

Aunque esta es una solución privada, permite el uso de algunos de sus componentes siempre que no sea para su uso en producción y con un mensaje informativo para adquirir una licencia.

En el proyecto se ha optado utilizar el paquete NuGet con la funcionalidad específica de visualizar mapas ofrecido por Syncfusion [28] dado que las mayorías de las soluciones para la representación de mapas están desarrolladas para JavaScript. En la Figura 6 se muestra un ejemplo de un mapa en la página web.

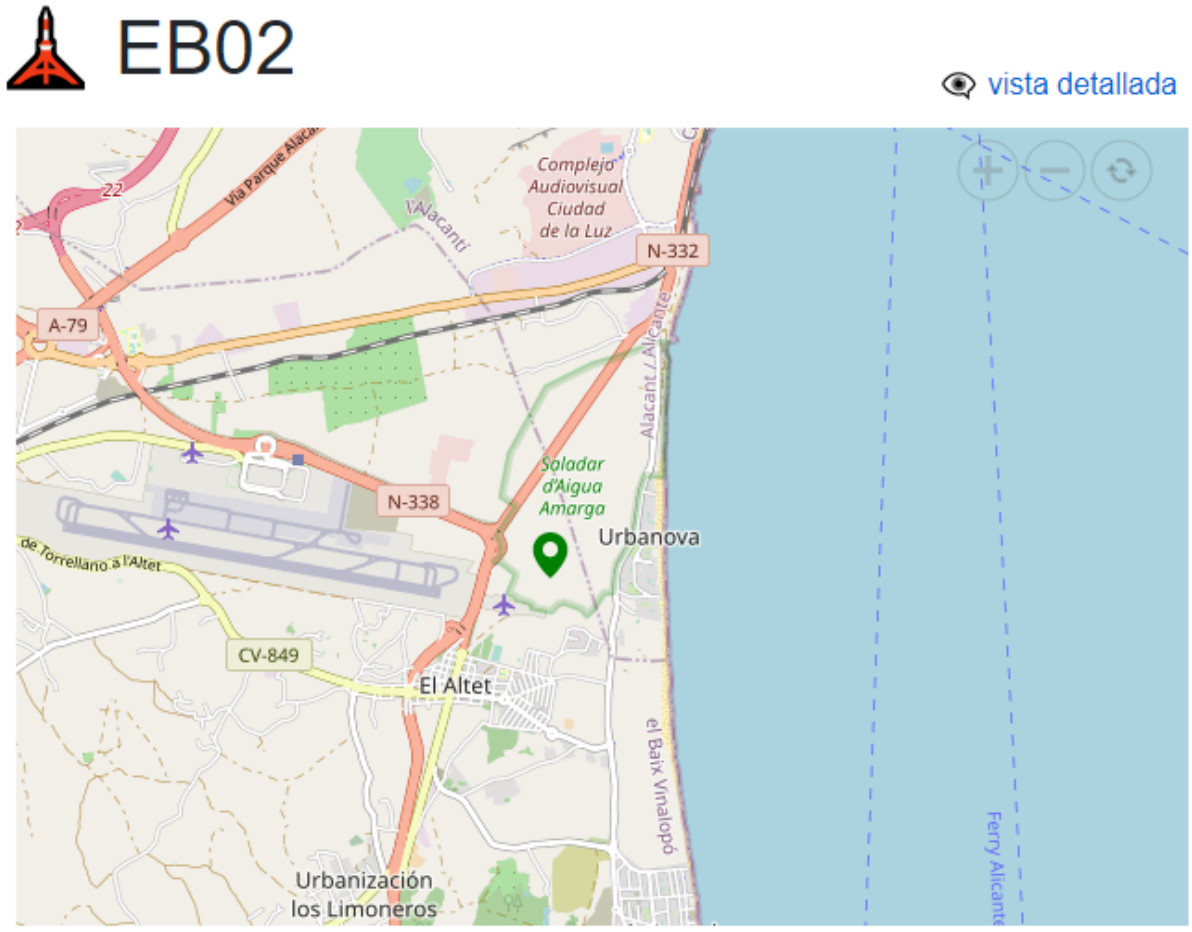


Figura 6. Localización de un sensor en el mapa

3.5.2 Blazorise

Blazorise [29] es una biblioteca de componentes de interfaz de usuario para Blazor que permite tener una abstracción sobre diferentes marcos de estilo CSS (Bootstrap, Material o AntDesign) de código abierto[15].

Este paquete se ha utilizado para la representación de las gráficas y mejorar la visualización de la información en el proyecto [30].

En la Figura 7 se muestra un ejemplo de las gráficas que muestran la información en la página web.

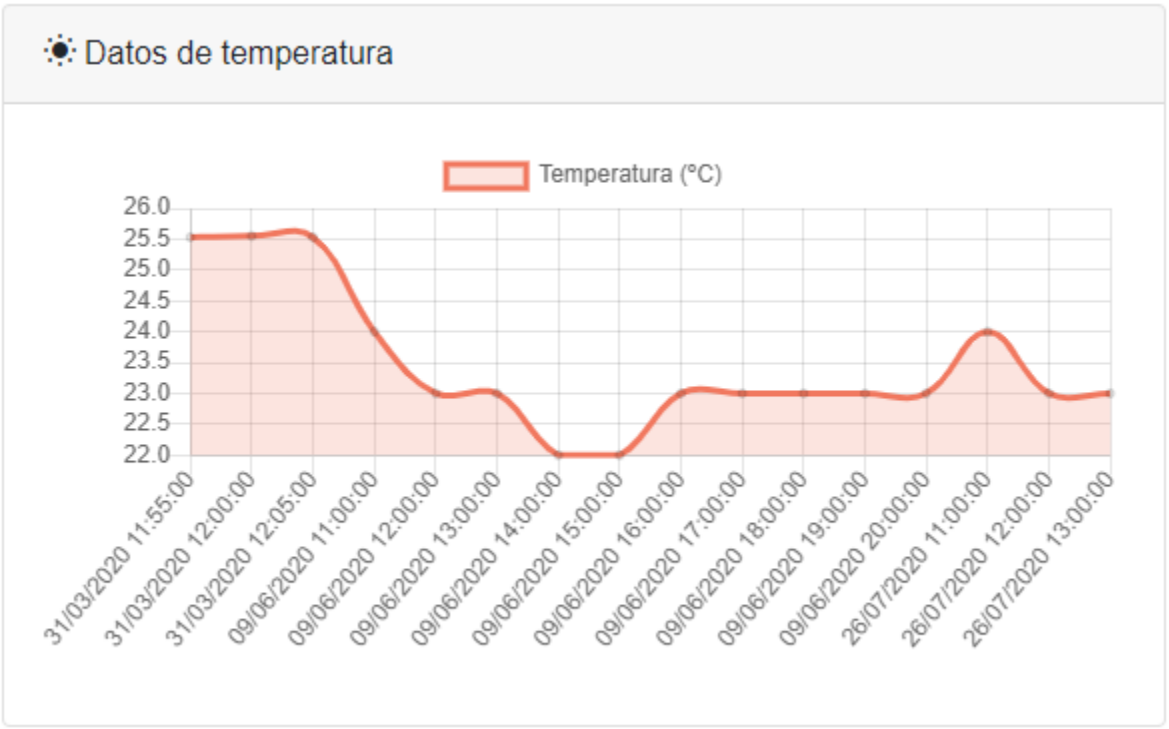


Figura 7. Datos de temperatura representados en una gráfica lineal

[15] https://github.com/stsrki/blazorise

4 Arquitectura de la plataforma

Durante el proceso de análisis de la arquitectura se plantearon distintas posibilidades tecnológicas para cada uno de los componentes de la misma. A continuación, se resumen las decisiones tomadas y tecnologías escogidas; más adelante en el documento se desarrollan con más profundidad.

Gran parte del proyecto se ha implementado haciendo uso de tecnologías de Microsoft: lenguaje de programación .Net Core, aplicación web Blazor, base de datos en SQL Server, Visual Studio como aplicación principal de desarrollo, etc. Sin embargo, no ha sido necesario invertir en licenciamiento ya que todas las opciones seleccionadas permiten ser utilizadas de forma gratuita y en distintos sistemas operativos; los componentes antes mencionados pueden funcionar en Windows y Linux indistintamente. El objetivo principal era trabajar con un sistema unificado y sencillo que facilitase la implementación de distintos tipos de componentes (APIs, páginas webs, servicios…) bajo un mismo lenguaje y patrón de diseño.

La solución aplicada implementa una serie de componentes muy distintos. Para comenzar, se encuentra el servicio (o Worker) alojado en las estaciones base que se encarga de enviar la información registrada por los sensores hacia la API. Esta API es otro de los componentes principales de la plataforma y se encarga de procesar la información enviada por los Workers para consolidarla en la base de datos. Finalmente, estos datos son explotados a través del tercer componente principal de la plataforma: la aplicación Web. Esta hace uso de diferentes herramientas gráficas para facilitar la visualización de la información.

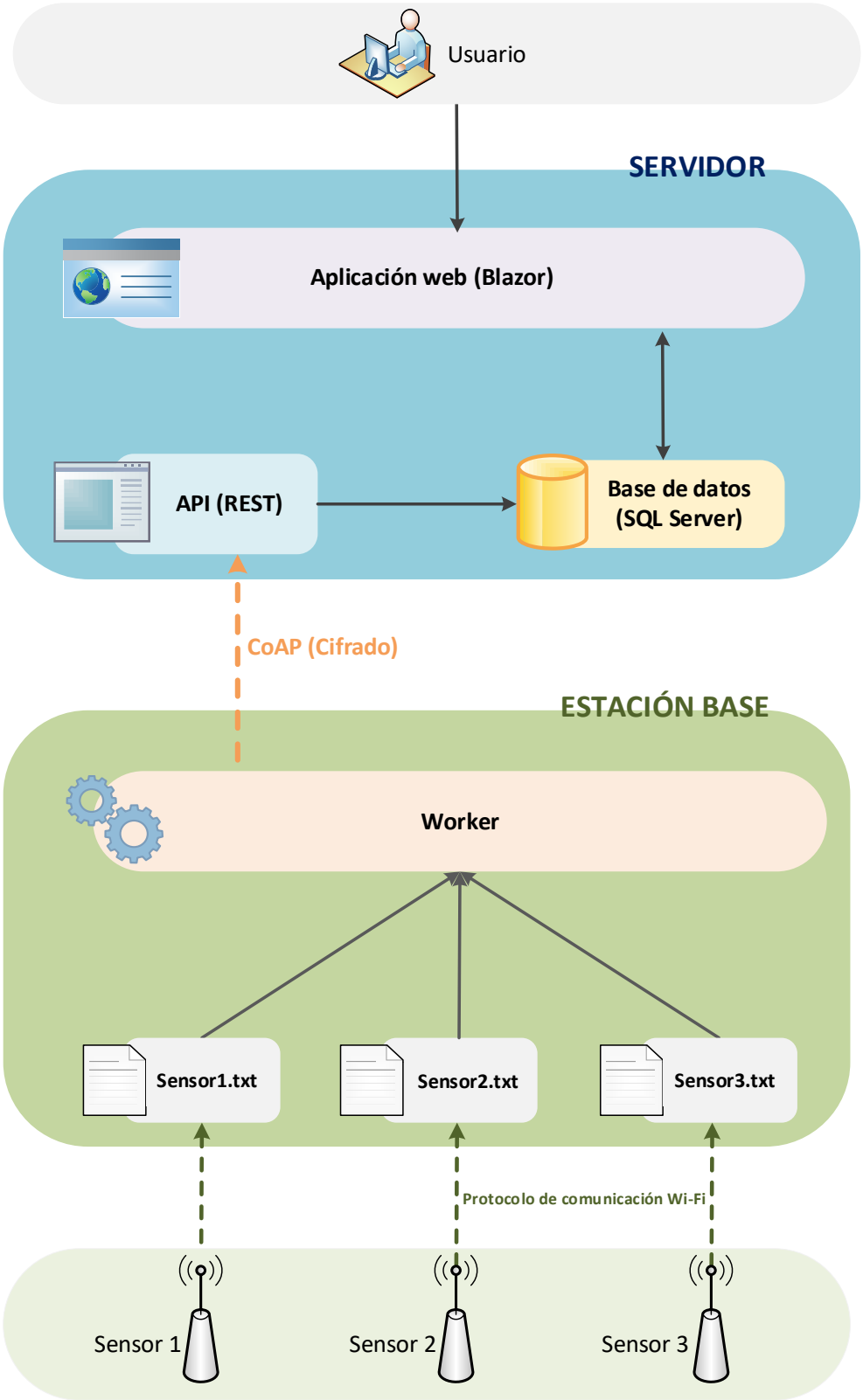


Figura 8. Arquitectura de la plataforma

Como se puede ver en la Figura 8, la plataforma hace uso de un servidor propio donde se han publicado la API y la aplicación web Blazor. Tanto la API como la aplicación web hacen uso de .NET Core, solución multiplataforma de Microsoft y, por tanto, el servidor web puede estar implementado tanto en un sistema Linux como en un sistema Windows.

Para la persistencia de datos (también alojado en el propio servidor) se ha hecho uso de SQL Server Express 2019. Esta solución es la versión gratuita de Microsoft que permite su uso en el desarrollo de aplicaciones web o de escritorio para producción y que tiene su versión multiplataforma desde 2017. Esta opción, además, está bien integrada con las herramientas Microsoft. En esta base de datos se consolidan los datos enviados a la API a través de los servicios Worker para que pueda ser consultada a través de la aplicación web.

La página web se ha construido como una aplicación Blazor ya que permite una mejor integración y un desarrollo más ágil al utilizar el mismo lenguaje (C#) que la parte del *back-end*. Además, Visual Studio provee de una serie de implementaciones base que ayudan a acelerar el desarrollo de la web. Algunas de esas implementaciones base son el *login* de usuario y el desarrollo base (menú y encabezados) de la página aplicando Bootstrap (consiguiendo así que la web sea *responsive*).

Para la comunicación entre los sensores y la estación base se hace uso de un protocolo propietario de la universidad de Alicante, mientras que para la comunicación entre las estaciones base ("Workert" en la Figura 8) y la API se ha hecho uso de un protocolo de comunicación IoT (CoAP); esta comunicación, a su vez, ha sido securizada haciendo uso de algoritmos de cifrado ampliamente utilizados hoy en día. El objetivo principal es conseguir afianzar una comunicación fiable en un entorno donde los dispositivos involucrados presentan recursos hardware limitados y donde el ancho de banda es muy reducido. La comunidad de Net Core ha facilitado esta tarea ya que el protocolo de comunicación empleado se encontraba ya implementado como una librería de código abierto.

La información recolectada por los sensores se escribe en formato JSON en los ficheros txt alojados en la estación base. El servicio antes comentado es el encargado de deserializar los datos de los archivos y enviarlos al servidor a través de peticiones realizadas a la API.

4.1 Diseño de base de datos

De manera general, la plataforma está compuesta principalmente por proyectos IoT. Cada proyecto, a su vez, se compone de una o más estaciones base que contienen un conjunto de sensores.

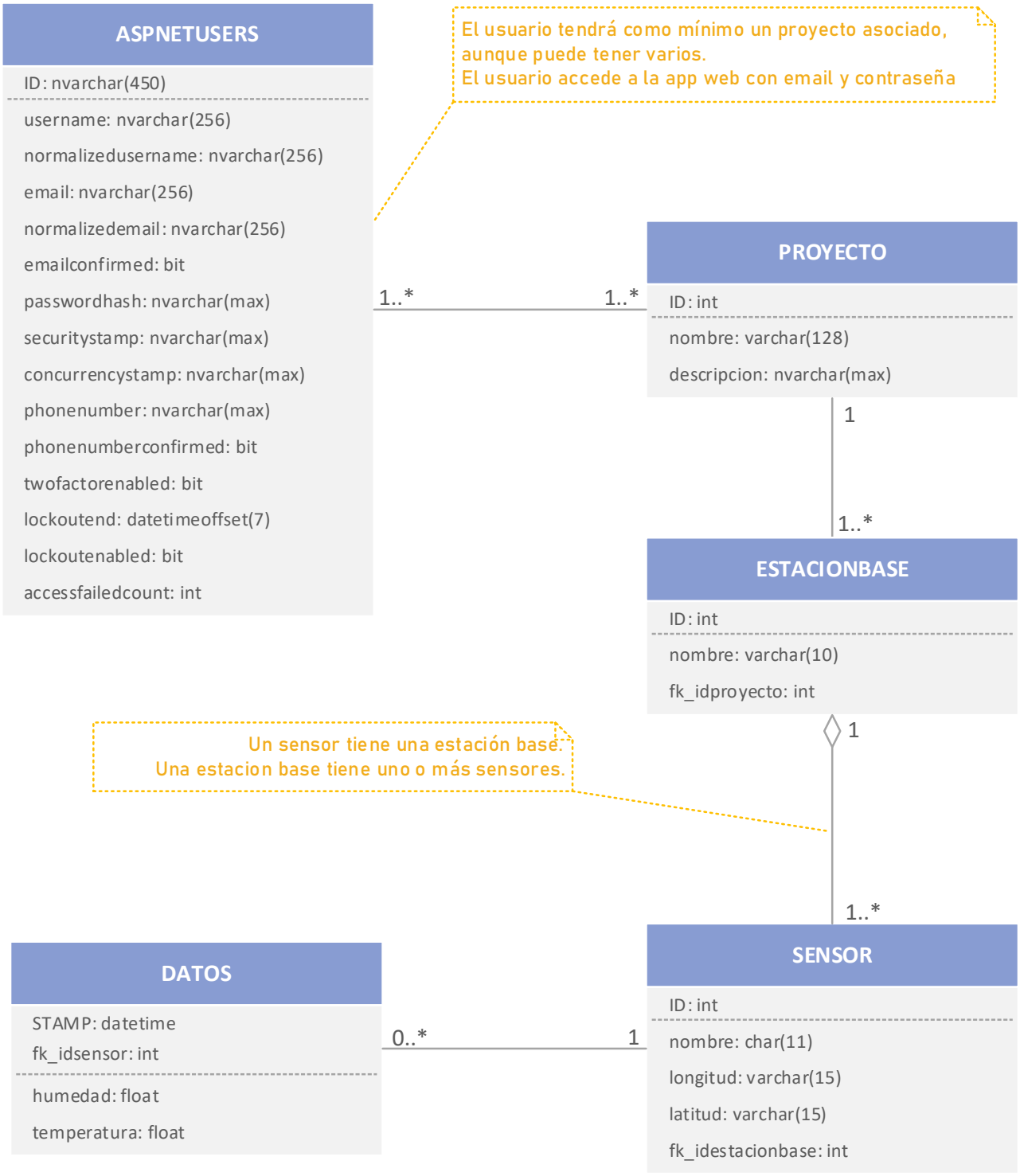


Figura 9. Esquema de la base de datos

En la tabla “Proyecto” se almacena la información relativa a los proyectos. En este caso, solamente se dispone de un identificador de tipo int identity, el nombre del proyecto y una descripción genérica de este.

La tabla de “EstacionBase” contiene el identificador (al igual que la tabla anterior), el nombre concreto de la estación y una referencia al proyecto al que pertenece. Como se puede ver en la Figura 9, un proyecto debe estar formado como mínimo por una estación base, pero puede tener más y una estación base va a pertenecer a un único proyecto. De esta forma, en la base de datos puede aparecer el mismo nombre de estación base repetido siempre que pertenezcan a proyectos diferentes.

La información relativa a los sensores se guarda en dos tablas. Por un lado, la tabla “Sensor” contiene información básica del sensor como es el nombre y las coordenadas (longitud y latitud) donde se encuentra y una referencia al proyecto al que pertenece (fk_idproyecto). Además, como se muestra en la figura 9, el identificador en este caso es de tipo int identity como en la tabla de proyectos. Por otro lado, la tabla “Datos” almacena la información relativa a cada lectura: el instante de tiempo en el que se ha realizado dicha lectura (*stamp*) y los valores de temperatura y humedad. Para relacionar los datos con el sensor se añade una referencia al sensor al que pertenece cada lectura (fk_idsensor).

En la Figura 9 se aprecia que la referencia al sensor en la tabla “Datos” forma parte de la clave primaria de la entidad. Esto posibilita tener lecturas en el mismo instante de tiempo para sensores diferentes.

Además, cada proyecto debe estar asociado a uno o más usuarios. En el caso del usuario se ha utilizado la tabla que ofrece la implementación base de Visual Studio para la autenticación de usuarios “Aspnetusers”.

En todas las tablas excepto en la de usuarios se utiliza el tipo int identity como clave primaria para facilitar y ganar velocidad en las consultas.

4.2 Arquitectura de la implementación

Para el desarrollo del proyecto se ha seguido una arquitectura en capas. Como se ve en la Figura 10, la capa de presentación (página web Blazor) y las aplicaciones (API y Worker) acceden únicamente a la capa de acceso a datos a través de la capa de negocio, la cual contiene la lógica principal de la aplicación y la definición de las entidades. En la capa de presentación, la aplicación web accede a la capa de dominio siguiendo el patrón MVVM.

Además, existe una capa común de soporte para todas las capas donde tiene lugar la gestión de log de errores.

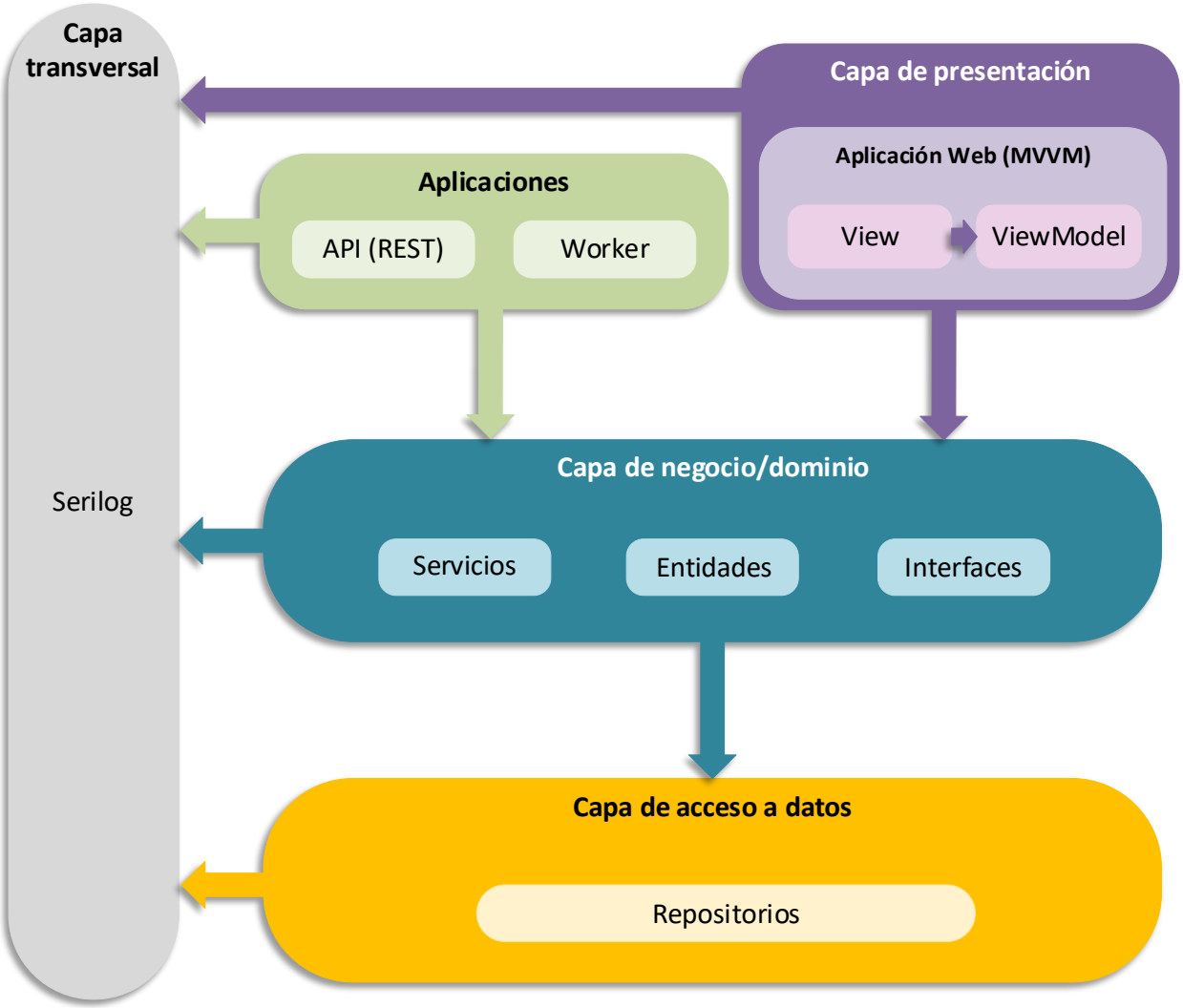


Figura 10. Arquitectura en capas de la implementación

Dentro de la capa de dominio se encuentran los servicios, las entidades y las interfaces:

- Los servicios agrupan el código referente a la lógica de negocio (cifrado/descifrado, acceso a repositorios, autorización de usuarios a recursos…).
- Las entidades son aquellas clases de código que únicamente representan objetos de la base de datos, es decir, no implementan ninguna función, solo definen atributos.
- Las interfaces son utilizadas para definir los métodos a implementar en los servicios y repositorios.

Dentro de la capa de acceso a datos se definen los repositorios. Éstos definen el código de acceso a la base de datos. Para evitar vulnerabilidades por inyección de SQL se ha hecho uso de un micro-ORM ampliamente utilizado: Dapper.

4.2.1 Worker

El worker es el servicio encargado de enviar a la API (a través de CoAP) la información de un conjunto de sensores asociados a una estación base. Este servicio se instala directamente sobre la estación base como un demonio Linux y se configura a través de un fichero JSON (cadena de conexión a la API, directorio del log de errores, clave pública del servidor…).

La información es almacenada por sensor, es decir, cada sensor envía la información a la estación base y ésta la almacena en un fichero en formato JSON. Cada fichero se corresponde con un único sensor y tiene como nombre el identificador del mismo. Además, el fichero de configuración del servicio también contiene el identificador de la estación base y del proyecto al que está asociada, así como el directorio donde se almacenan los ficheros de los sensores.

El worker es el encargado de leer los datos de cada uno de los ficheros existentes en la estación base y los datos del archivo de configuración, formar una petición que pueda ser atendida por la API, cifrarla mediante una metodología que se explicará más adelante y enviarla a través del protocolo CoAP.

4.2.1.1 Ficheros de datos

En cada fichero se almacena un conjunto de datos de un sensor; cada captura de datos se inserta como una nueva línea dentro del fichero. Cada línea presenta datos de temperatura y humedad para un instante de tiempo concreto.

La información se escribe en el fichero en formato JSON según el estilo establecido que se puede ver a continuación:

```
{"Stamp": "2020-06-09 00:00:00", "Humedad": "10.00", "Temperatura": "16.00"}
```

A continuación, en la Figura 11, se muestra un fichero completo de ejemplo.

```
{ "Stamp": "2020-06-09 00:00:00", "Humedad": "10", "Temperatura": "16.00"}
{ "Stamp": "2020-06-09 02:00:00", "Humedad": "10", "Temperatura": "16.00"}
{ "Stamp": "2020-06-09 03:00:00", "Humedad": "10", "Temperatura": "16.00"}
{ "Stamp": "2020-06-09 04:00:00", "Humedad": "10", "Temperatura": "16.00"}
{ "Stamp": "2020-06-09 05:00:00", "Humedad": "10", "Temperatura": "15.00"}
{ "Stamp": "2020-06-09 06:00:00", "Humedad": "10", "Temperatura": "16.00"}
{ "Stamp": "2020-06-09 07:00:00", "Humedad": "10", "Temperatura": "17.00"}
{ "Stamp": "2020-06-09 08:00:00", "Humedad": "10", "Temperatura": "18.00"}
{ "Stamp": "2020-06-09 09:00:00", "Humedad": "0", "Temperatura": "20.00"}
{ "Stamp": "2020-06-09 10:00:00", "Humedad": "0", "Temperatura": "22.00"}
{ "Stamp": "2020-06-09 11:00:00", "Humedad": "10", "Temperatura": "24.00"}
{ "Stamp": "2020-06-09 12:00:00", "Humedad": "10", "Temperatura": "23.00"}
```

Figura 11. Fichero de texto con datos emulados

4.2.2 API (REST)

La API es un servicio web alojado en el servidor central y que está preparado para recibir peticiones de tipo POST a través del protocolo CoAP. Al igual que el Worker, se puede configurar a través de un fichero JSON donde se especifica, entre otras cosas, el directorio de la clave privada RSA, la cadena de conexión con la base de datos y el directorio del log de errores.

La API se encarga de descifrar la petición recibida e insertar los datos en el servidor. Si el servidor procesa correctamente la petición devuelve el código 2.04 Changed al cliente; en caso contrario, si no puede interpretar la petición devuelve el código 4.00 Bad Request (ver detalle en el apartado 3.1.2). El servidor solo acepta el envío de datos a través de peticiones POST y no necesita devolver ningún recurso al cliente, por lo tanto, el código 2.04 Changed es la mejor opción en comparación al código 2.05 Content.

4.2.3 Aplicación Web

La página web desarrollada en Blazor se utiliza principalmente para la visualización de los datos recibidos por la API. También se utiliza para dar de alta los proyectos, estaciones base y sensores de los cuales se consolida la información. La web hace uso de gráficas para mejorar la visualización de los datos recabados de los sensores y también utiliza los mapas para poder mostrar la localización de cada uno de ellos.

Para la implementación de la página web se ha seguido el patrón MVVM (ver apartado 3.2.1) con el objetivo de desacoplar aún más la capa de vista. Al mismo tiempo, también se ha seguido el patrón por capas en el Worker y la API para poder consumir los recursos almacenados: la capa viewmodel consume los recursos de la capa de servicio para poder leer/editar la información en la base de datos. De esta forma el proyecto web hace uso de ambos patrones con el objetivo de conseguir un buen desacoplamiento en el *front-end* y una implementación uniforme (con respecto al proyecto completo) en el *back-end*.

La página web está securizada con un login de manera que solo los usuarios autenticados pueden acceder y, además, tan solo permite al usuario visualizar los datos a los que tiene permiso.

4.3 Mecanismos de seguridad

En la solución, la estación base (un demonio instalado en una máquina virtual a modo de simulación) envía los datos que recauda de los sensores al servidor. Para ello, hace una consulta de tipo POST a una API configurada en dicho servidor a través del protocolo de comunicación para dispositivos IoT CoAP.

El protocolo CoAP que se ha utilizado sigue el estándar RFC 7252; este estándar además define un sistema de comunicación seguro (denominado CoAPS) que permite el envío de información cifrada.

Para el desarrollo se ha utilizado un paquete *NuGet* en ASP.NET Core de código abierto que implementa CoAP siguiendo el estándar RFC 7252; sin embargo, esta librería no implementa el protocolo CoAPS y, por lo tanto, ha sido necesario implementar un mecanismo de comunicación seguro sobre CoAP manualmente.

4.3.1 Comunicación segura Worker-API

Para implementar el mecanismo de comunicación segura se ha hecho uso de los algoritmos previamente mencionados (AES y RSA). Se ha utilizado la implementación de dichos algoritmos ofrecida por librerías nativas de ASP .NET, por lo que se ha asumido que es correcta, estándar y ampliamente utilizada.

Se trata de una comunicación unidireccional estación base – servidor, por lo que inicialmente se planteó utiliza solamente el algoritmo de cifrado RSA. En este caso, el servidor dispone de un par de claves pública/privada, la estación base solamente conoce la clave pública del servidor y la utiliza para cifrar la información que debe enviar. El servidor con la clave privada (que solo él conoce) descifra el mensaje recibido.

Al implementarlo y hacer pruebas se pudo comprobar que dicha solución no es viable para la aplicación desarrollada puesto que la cantidad de datos a cifrar puede llegar a ser demasiado grande y esto no resulta compatible con el algoritmo (aun haciendo uso de claves de cifrado de 4096 bits).

Como solución a esta problemática se ha optado por utilizar una combinación de AES y RSA.

La información que se va a enviar (los datos captados por los sensores) se cifra utilizando el algoritmo AES. La clave y el vector de inicialización de AES son generados por la estación base con cada nuevo envío de datos. Esta clave y su vector de inicialización son cifrados con la clave pública (algoritmo RSA) del servidor y son enviados junto con la información de los sensores al servidor (Figura 12. Cifrado en estación base). De esta forma, se puede enviar gran cantidad de información cifrada sin necesidad de tener que crear un proceso de generación y renovación de las claves AES entre la estación base y el servidor.

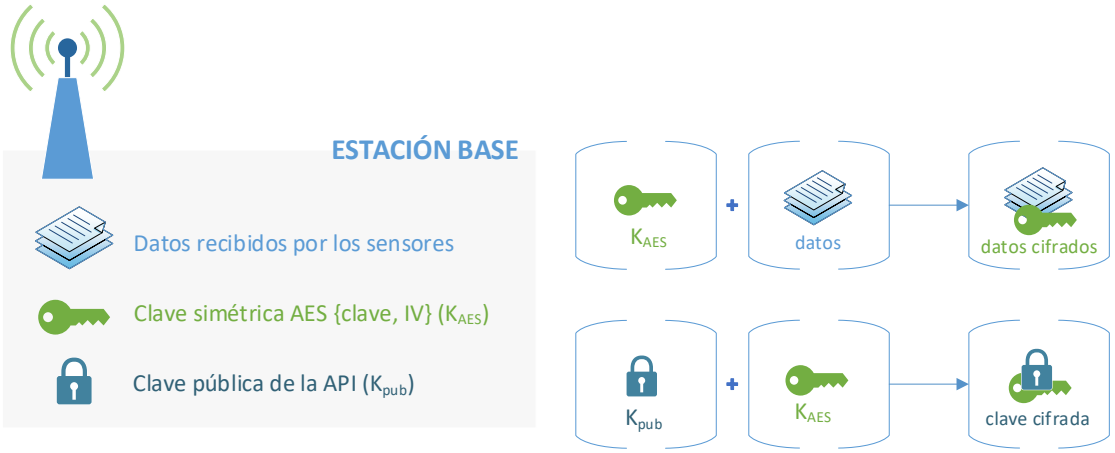


Figura 12. Cifrado en estación base

En este escenario, la clave pública no se comparte a través de la red, sino que es el administrador el que conoce la clave y la añade en la configuración de las estaciones base que deben enviar información. De esta forma, tan solo las estaciones base configuradas por este administrador son las que pueden enviar información cifrada al servidor.

En el lado del servidor se descifra la clave simétrica haciendo uso de la clave privada de este. Una vez obtenida la clave con la que se ha cifrado la información, se descifran los datos enviados por la estación base (con la clave simétrica) para almacenarlos posteriormente en la base de datos (Figura 13. Descifrado en API).

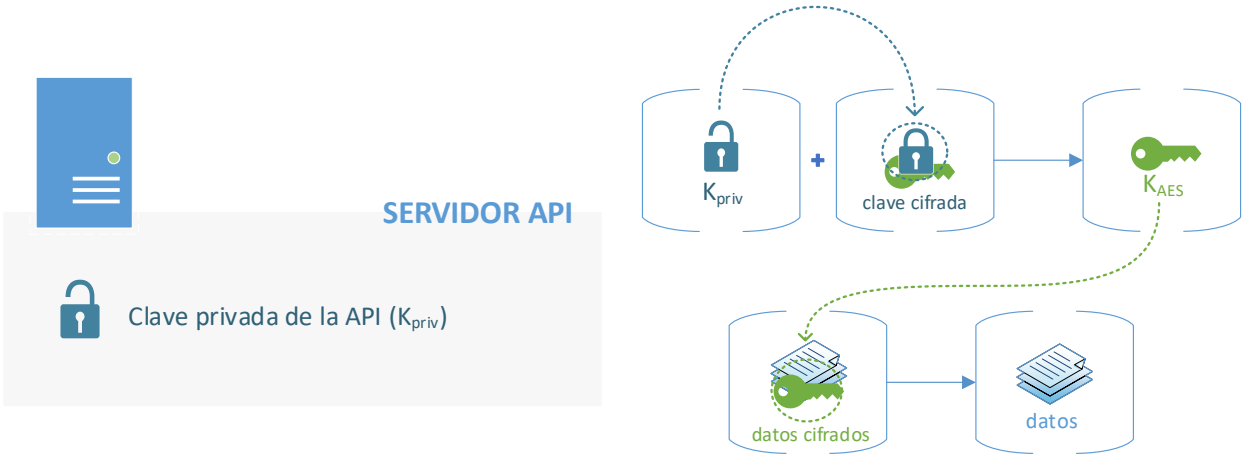


Figura 13. Descifrado en API

4.3.2 Diagrama resumen de la comunicación segura

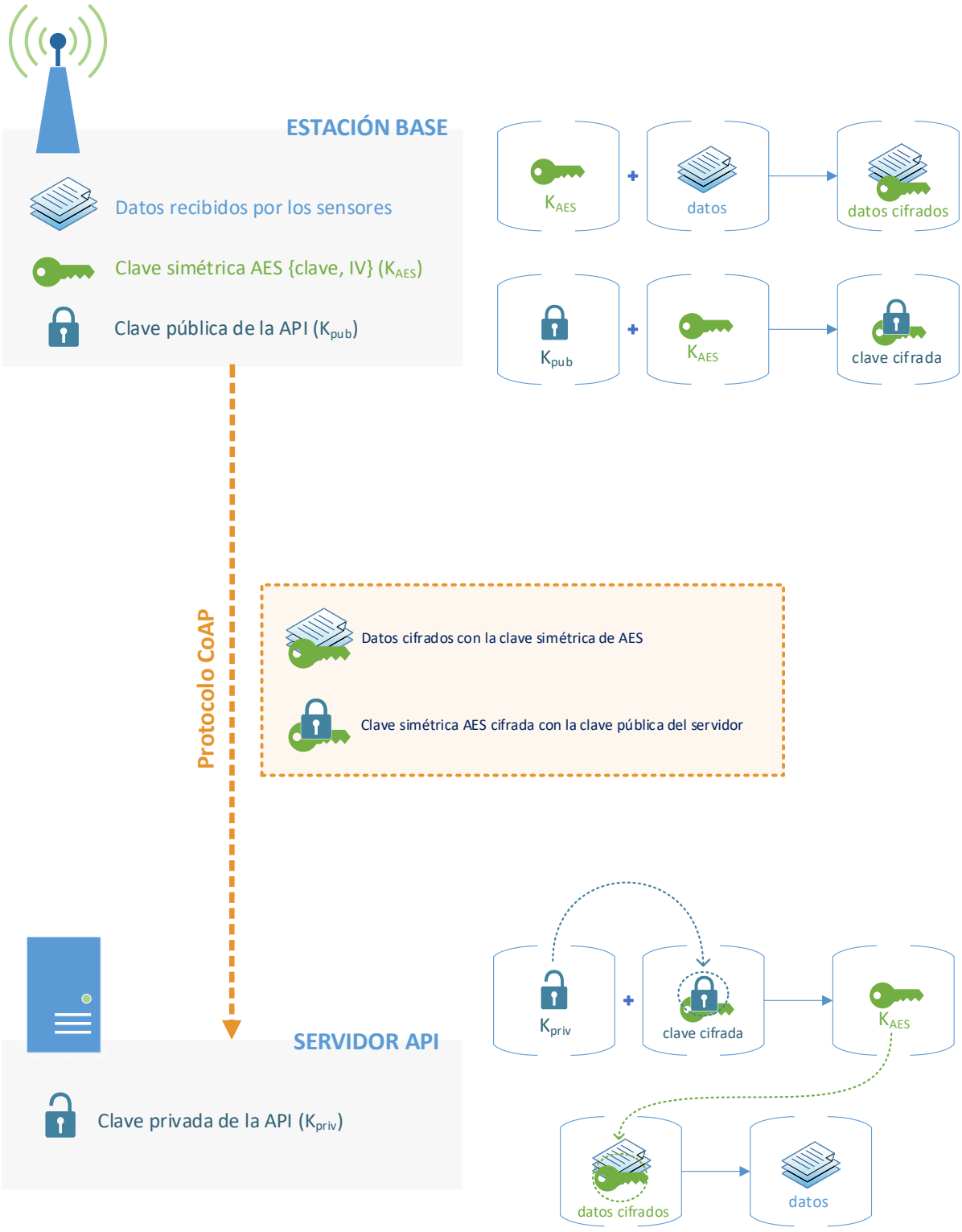


Figura 14. Comunicación segura

4.4 Diseño de la aplicación web

Antes de implementar la página web es necesario realizar un esbozo de esta para tener claro cómo va a interactuar el usuario con la página y qué información se va a mostrar; de esta manera se tiene una primera visión básica de cómo se va a distribuir la aplicación, qué información se va a mostrar y cómo.

Además, realizar un diseño previo a través de mockups ayuda a que la página sea *user-friendly* sin necesidad de tener que estar haciendo y deshaciendo constantemente la implementación realizada.

A continuación, se muestran los bocetos realizados para la aplicación web. Estos bocetos se han realizado con la herramienta Moqups que ofrece un plan gratuito para uso básico.

En primer lugar, se muestra la idea preliminar para la visualización de proyectos y estaciones base que era mostrar cada proyecto con sus estaciones asociadas es una única página (Figura 15. Mockup de la vista de proyectos).

Figura 15. Mockup de la vista de proyectos

Inicialmente, para las estaciones basé se planteó donde mostrar los sensores asociados a cada una de ellas (Figura 16. Mockup de la vista de una estación base).

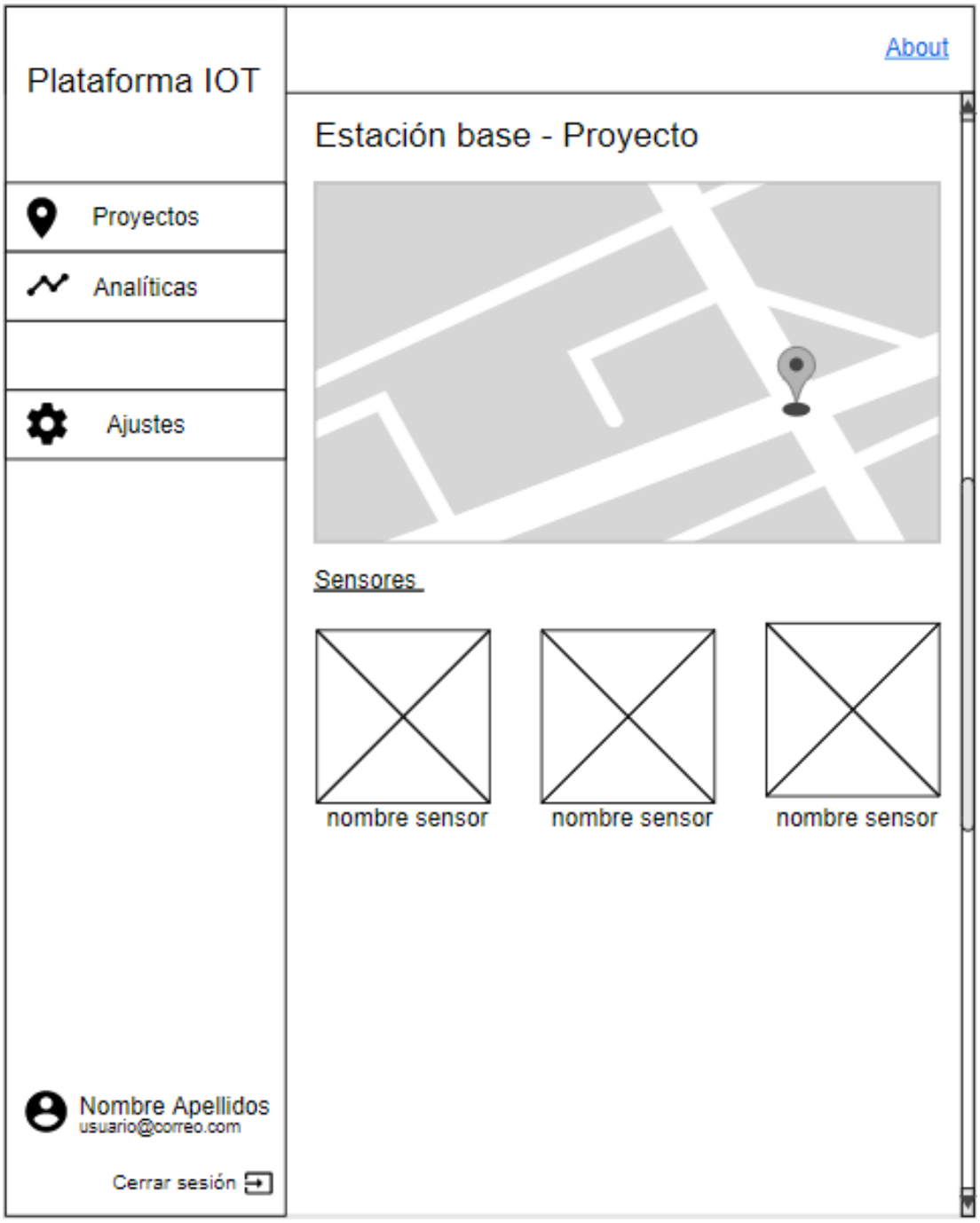


Figura 16. Mockup de la vista de una estación base

Finalmente, en la vista que se planteó para cada sensor se muestra la información captada por los mismos (Figura 17. Mockup de la vista de un sensor).

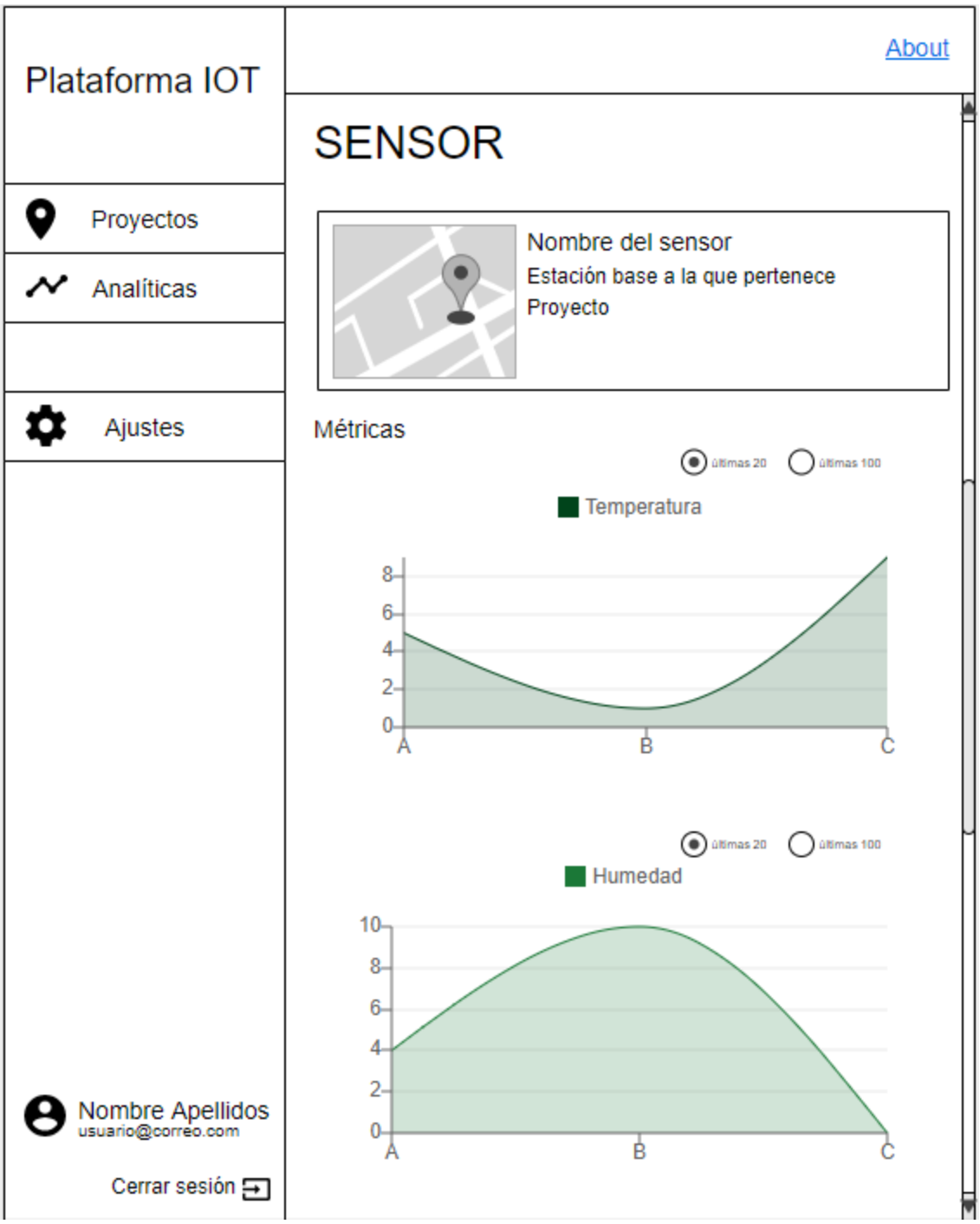


Figura 17. Mockup de la vista de un sensor

Las funcionalidades de la aplicación se explican más detalladamente más adelante en el documento (ver apartado 5.4)

5 Implementación de la plataforma

El proyecto, como se ha visto en el apartado de la arquitectura, consta de tres elementos básicos: el worker, la API y la página web. En los apartados siguientes se comenta las funcionalidades básicas de cada uno de estos elementos y se muestra el pseudocódigo de algunos de los métodos más relevantes.

5.1 Worker en estación base (cliente)

Como ya se ha comentado, el worker se encarga de leer los ficheros de datos de la estación base y enviarla de forma segura a la API.

El worker se ejecuta cada minuto para leer los archivos que ha creado/rellenado la estación base con los datos de los sensores para formar la petición (una por archivo) y enviarla al servidor. Cuando envía los datos de un fichero lo elimina; tanto si funciona correctamente la inserción como si no, el fichero es eliminado. El principal motivo por el que se eliminan los ficheros que no se han podido insertar correctamente es para evitar que el disco duro de la estación base se llene si hay numerosos ficheros con datos corruptos, incorrectos o pertenecientes a sensores aún no dados de alta para la estación base en la base de datos a través de la aplicación web. En caso de error se añade un mensaje descriptivo en el log de la API indicando el motivo de la inserción fallida.

A continuación, se muestra el pseudocódigo seguido para la lectura de ficheros en la estación base y el envío de datos al servidor. Este es el proceso se ejecuta en el intervalo de tiempo establecido en el fichero de configuración (el mínimo tiempo de refresco es un minuto para evitar sobrecargar demasiado el worker).

Envío de datos:

```
Inicio
      para cada archivo en estacion_base entonces
            peticion <- ObtenerDatos(archivo);
            respuesta <- Enviar(peticion);
            si respuesta == "Changed" entonces
                  EscribirLog("Información insertada correctamente.");
                  Eliminar(archivo);
            si no entonces
                  EscribirLog("Fallo al insertar la información.");
                  Eliminar(archivo);
            fin si
      fin para cada
Fin
```

Obtención de datos:

```
Inicio
      Lista datos;
      AbrirArchivo(archivo);
      Mientras queden líneas entonces
            linea <- LeerLinea(archivo);
            dato <- Deserializar(linea);
            datos += dato;
      Fin mientras
      peticion <- CrearPeticion(datos);
      peticionSegura <- Cifrar(peticion);
      Devolver(Serializar(peticionSegura));
Fin
```

El método que se ha utilizado para cifrar la petición se comenta más adelante en el documento.

5.2 API (REST) en servidor

En apartados anteriores se ha comentado que la API es la intermediaria entre el worker y la base de datos: procesa todas las peticiones procedentes de los workers y las inserta en la base de datos.

A continuación, se muestra el pseudocódigo que describe el procedimiento seguido por la API al recibir una petición a través del protocolo CoAP.

```
Inicio
      peticionRecibida;
      si (peticionRecibida != null) entonces
            peticionSegura <- Deserializar(peticionRecibida);
            peticion <- Descifrar(peticionSegura);
            resultado <- InsertarPeticion(peticion);
            si (resultado == true) entonces
                  Devolver(Changed);
            si no
                  Devolver(BadRequest);
            fin si
      si no
            EscribirLog("La petición recibida está vacía.");
      Fin si
Fin
```

El método utilizado para descifrar se explica en el apartado 5.3. Los métodos de descifrado comentados más adelante hacen uso de la clave privada del servidor. Para tener acceso a dicha clave se debe configurar el directorio donde está almacenada en el fichero de configuración "appsettings.json" de la API.

A continuación, se muestra el fichero con datos de ejemplo (Figura 18).

Figura 18. Fichero de configuración de la API

5.3 Implementación de la comunicación segura (Worker-API)

En primer lugar, el worker en cada estación base lee los ficheros con la información captada por los sensores (un archivo por sensor) y deserializa dicha información para construir una petición que enviará a la API a través del protocolo CoAP (ver apartado 5.1). Cada línea de un fichero representa (en formato JSON) una muestra de temperatura y humedad en un momento concreto.

A continuación, la información leída del archivo es cifrada con la clave simétrica (AES) generada. Con la información cifrada y la clave AES se crea la petición que será enviada al servidor. La clave AES se cifra con la clave pública (RSA) de la API antes de ser incluida en la petición.

Cifrado:

```
Inicio
      peticion;
      peticionSerializada <- Serializar(peticion);
      claveAes <- GenerarClavesAES();
      peticionSerializadaSegura <- CifrarAES(peticionSerializada, claveAes);
      claveAesSegura <- CifrarRSA(claveAes);
      peticionSegura <- CrearPeticionSegura(peticionSerializadaSegura,
claveAesSegura);
      Devolver(peticionSegura);
Fin
```

Una vez que se ha creado la petición segura, el worker la envía a la API mediante una petición POST a través de CoAP.

La petición se deserializa al llegar a la API. Primero se descifra (utilizando la clave privada RSA) la clave AES y, posteriormente esta se utiliza para descifrar la información de los sensores [31].

Descifrado:

```
Inicio
      peticionSegura;
      claveAesSegura <- peticionSegura(claveAes);
      peticionSerializadaSegura <- peticionSegura(peticionSerializada);
      claveAes <- DescifrarRSA(claveAesSegura);
      peticionSerilizada <- DescifrarAES(peticionSerializadaSegura,
claveAes);
      peticion <- Deserializar(peticionSerilizada);
      Devolver(peticion);
Fin
```

Finalmente, los datos descifrados son insertados en la base de datos por la API (ver apartado 0).

Cabe destacar que con el objetivo de mejorar la legibilidad y mantenibilidad del código todos los algoritmos encargados de la seguridad (incluyendo el cifrado y descifrado de peticiones) han sido implementados en una única clase. De esta forma, y gracias al patrón de desarrollo utilizado, no es necesario que el worker o la API implementen código específico de seguridad ya que todo está contenido en la misma librería.

5.4 Aplicación web para la administración/visualización de la información

A continuación, se detallan las funcionalidades implementadas en la aplicación web. Muchas de ellas están orientadas a la visualización de datos, aunque también permite la edición y el borrado de los mismos.

Una de las funcionalidades básicas implementadas es el login. Los usuarios pueden registrarse y acceder a la web con un correo electrónico y eliminar todos sus datos cuando lo indiquen (Figura 19. Página de inicio de sesión).

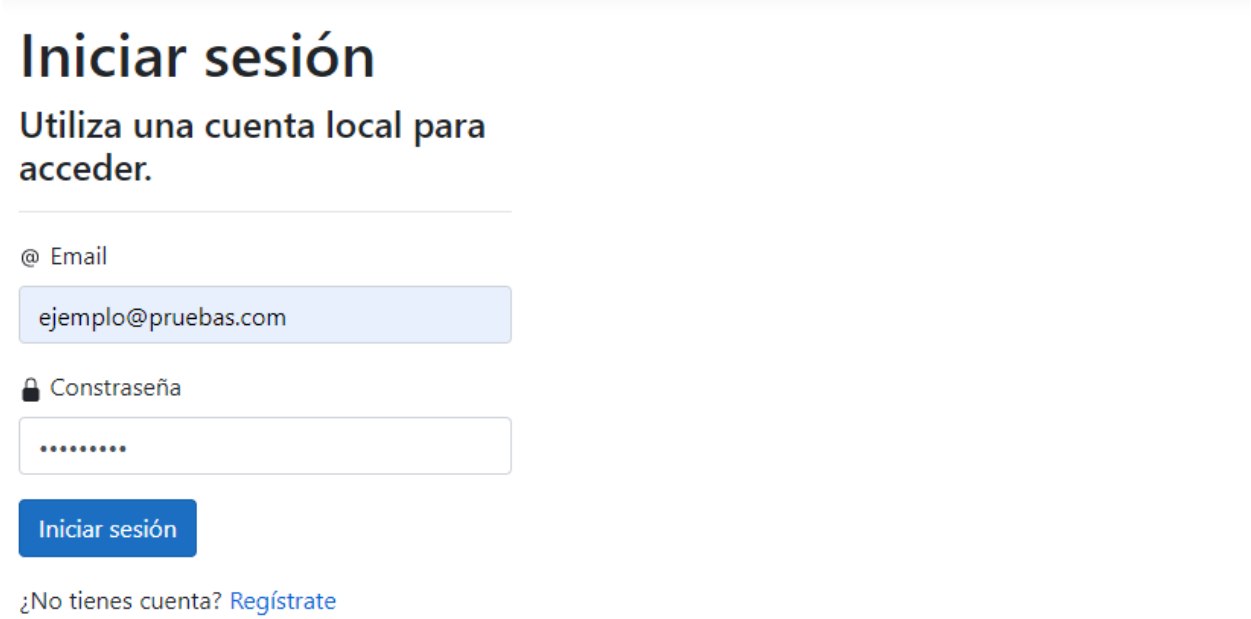


Figura 19. Página de inicio de sesión

La aplicación web permite añadir, editar y eliminar proyectos, estaciones base y sensores en la venta de administración (ver Figura 20. Ventana de administración de proyectos). Es importante señalar que hasta que estos datos no se han dado de alta a través de la aplicación web, la API no registra la información referente de ese proyecto, aunque reciba peticiones relacionadas con el mismo. Es necesario que se den de alta los sensores y estaciones base para que la API pueda registrar la información correctamente.

En la misma ventana, es posible generar un par de claves pública/privada (RSA) que pueden ser utilizadas para el cifrado entre los *workers* (estaciones base) y la API. Las claves se generan de manera aleatoria por la propia librería RSA de .NET Core y no se almacenan en el servidor. La web comprime las claves en un archivo zip para facilitar su descarga y posteriormente elimina todos los ficheros auxiliares utilizados; de esta forma, las claves solo están en posesión del usuario.

Figura 20. Ventana de administración de proyectos

Al mismo tiempo, la web permite visualizar toda la información referente a los proyectos. Se puede ver cuáles son sus estaciones base y sensores asociados (ver Figura 21. Vista de proyectos con estaciones base asociadas).

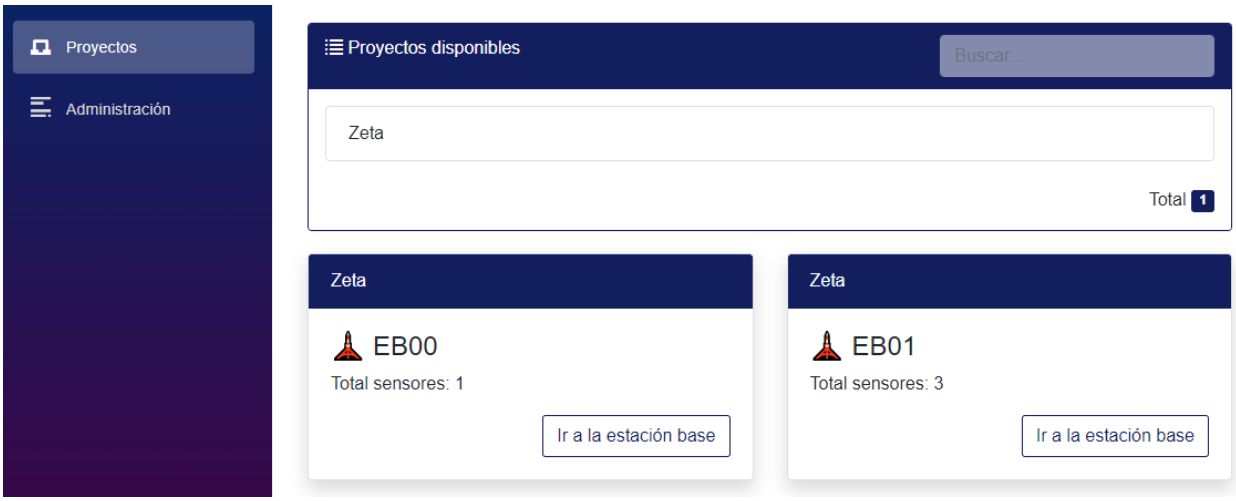


Figura 21. Vista de proyectos con estaciones base asociadas

Además, la página ofrece una información más detalla referente a los sensores y estaciones base:

- Último dato registrado (con su fecha) y geolocalización en mapa de los sensores (Figura 22. Vista de los sensores de una estación base)

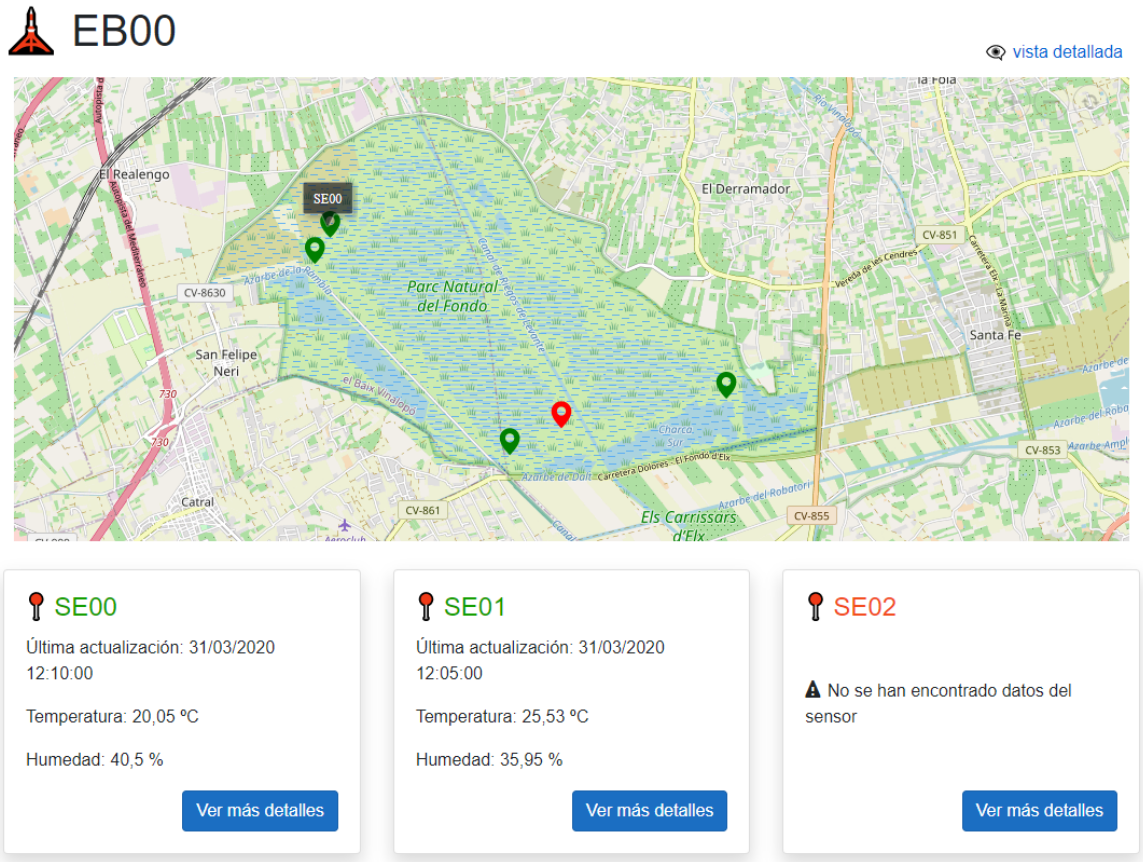
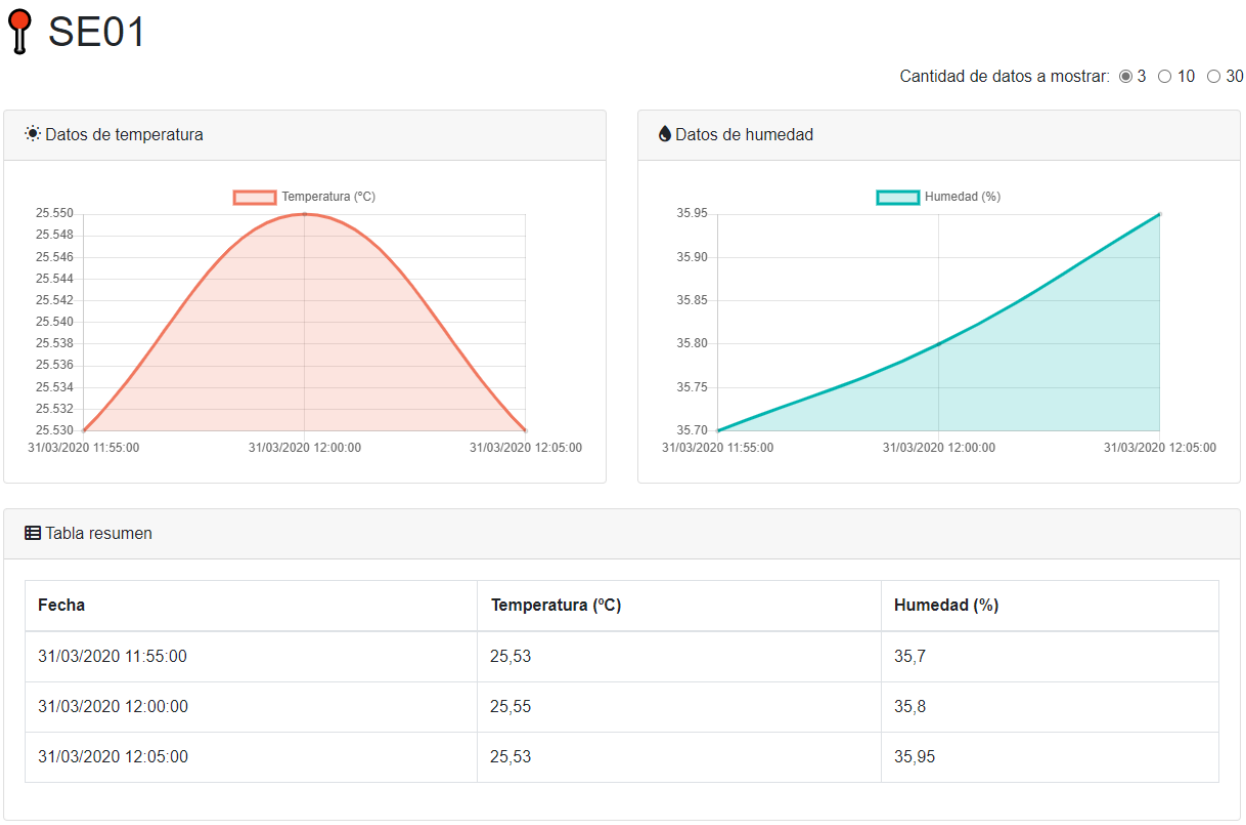


Figura 22. Vista de los sensores de una estación base

- Vista detallada de los últimos datos registrados para un sensor (Figura 23. Vista detallada de los datos de un sensor)



Fecha	Temperatura (ºC)	Humedad (%)
31/03/2020 11:55:00	25,53	35,7
31/03/2020 12:00:00	25,55	35,8
31/03/2020 12:05:00	25,53	35,95

Figura 23. Vista detallada de los datos de un sensor

- Vista comparativa en la que se muestran los datos de todos los sensores de una única estación base, estos datos pueden filtrarse por fecha (Figura 24. Comparativa de los datos de los sensores de una estación base)

Figura 24. Comparativa de los datos de los sensores de una estación base

A continuación, se muestra el pseudocódigo que se ha utilizado para implementar dos de las funciones más relevantes de la aplicación web ya que gran parte de los métodos implementados tan solo sirven para leer, insertar y actualizar datos de la base de datos o para tratar procesos concretos del *front-end* que no tienen tanta importancia.

El primer método que se muestra es el empleado para verificar que el usuario está autenticado y que tiene permisos para visualizar la información solicitada; este proceso de verificación tiene lugar cada vez que se solicitan recursos a la página web. Para poder replicar esta funcionalidad en todas las páginas de la web se ha aprovechado el patrón MVVM comentado anteriormente. El pseudocódigo que se muestra a continuación se ha implementado como una clase en la capa *ViewModel* y es el que utilizan todas las páginas (*Views*) para asegurar que el usuario está autenticado/autorizado.

Comprobación de autenticación/autorización de usuario:

```
Inicio
      usuario;
      autorizado;
      idUsuario;
      si (usuario no está autenticado) entonces
            autorizado <- false;
            Redirigir(Login);
      si (idUsuario es vacío) entonces
            idUsuario <- ObtenerId(usuario);
            autorizado <- true;
      fin si
      si (recurso no es vacío) entonces
            autorizado <- EstaAutorizado(idUsuario, recurso)
      fin si
Fin
```

La variable “autorizado” mostrada en el pseudocódigo anterior sirve para decidir si el usuario actual puede visualizar la información que está solicitando (es decir, está autenticado y tiene permisos).

Seguidamente, se muestra el método encargado de generar/comprimir el par de claves RSA en un directorio temporal y la limpieza de archivo auxiliares en el mismo.

Generación de claves RSA:

```
Inicio
      directorioTemporal;
      ficheroClavePublica;
      ficheroClavePrivada;
      zipClaves;
      zipClavesMemoria;

      ficheroClavePublica <- CrearClavePublica(directorioTemporal);
      ficheroClavePrivada <- CrearClavePrivada(directorioTemporal);
      zipClaves <- Comprimir(ficheroClavePublica, ficheroClavePrivada);
      zipClavesMemoria <- CargarFichero(zipClaves);
      Si (directorioTemporal != vacío)  entonces
            Eliminar(directorioTemporal);
      Fin si
      Si (zipClaves != vacio) entonces
            Eliminar(zipClaves);
      Fin si

      Devolver(zipClavesMemoria);
Fin
```

6 Pruebas y validación

Inicialmente el proyecto se planteó para realizar las pruebas de la plataforma en un entorno real, pero debido a la pandemia ocasionada por el Covid-19[16] no ha sido posible. En su lugar se han emulado datos representativos para mostrar y validar el correcto funcionamiento de la página web.

A continuación, se describe el proceso llevado a cabo en una prueba básica con un par de sensores para comprobar y validar el correcto funcionamiento de la plataforma (el sistema ha sido probado con más sensores de manera adicional). Además, en este apartado se pretende dar a conocer más en profundidad cada una de las funciones de la página y ver cómo funcionan.

Antes de comenzar a enviar información desde la estación base, es necesario prepararla. Al final del documento se ha añadido un anexo donde se explica el despliegue del servicio worker en la estación base.

6.1 Registro y login de usuario

6.1.1 Crear una nueva cuenta

El acceso a la página está restringido sólo para usuarios que tengan cuenta en el sistema, de manera que es necesario crear una para realizar las pruebas.

Para crear una nueva cuenta solo es necesario introducir un correo electrónico y una contraseña de acceso que debe contener al menos 6 caracteres entre los que se incluya, al menos, una mayúscula, una minúscula, un número y un carácter especial.

En la Figura 25 se muestra el aspecto del formulario de registro del usuario utilizado en la fase de experimentación.

[16] https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019

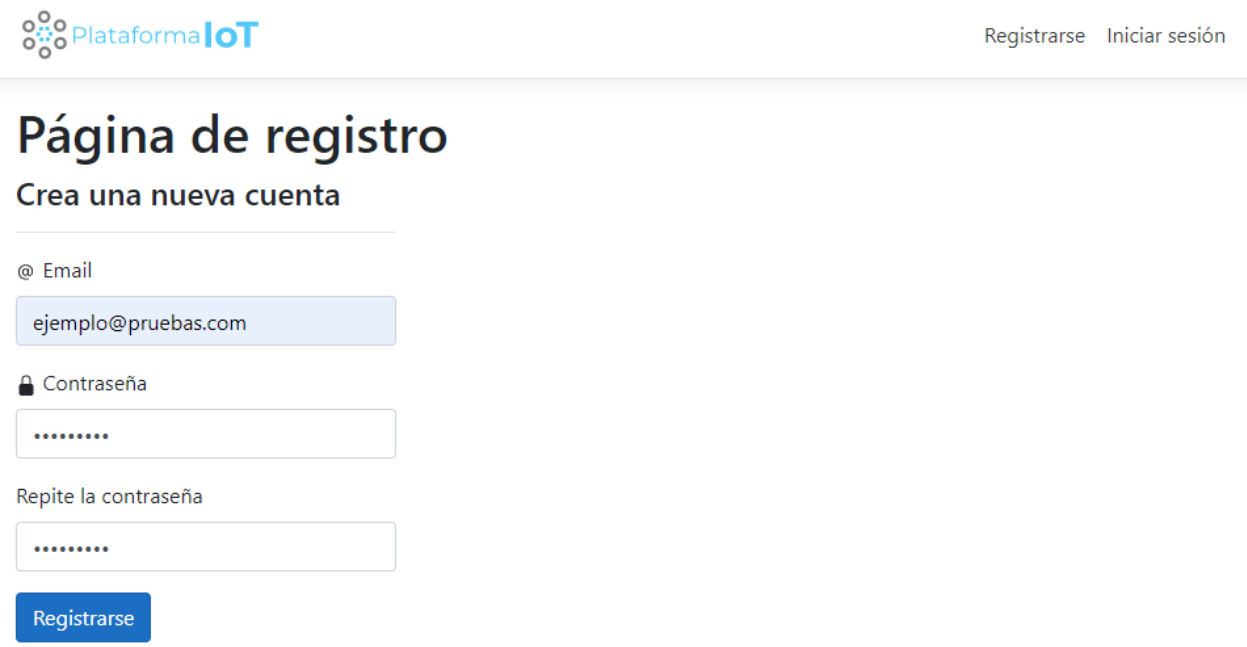


Figura 25. Página de registro para nuevos usuarios

El acceso dicho formulario se encuentra en la página de bienvenida. En la Figura 26 se marcan los enlaces que llevan a dicho formulario.

6.1.2 Acceder a la página web

La página de *login* (ver Figura 26) es la primera que se muestra al acceder a la página ya que es necesario estar autenticado para poder acceder al resto de pantallas de la aplicación (si el usuario no está autenticado no tendrá acceso, aunque introduzca manualmente la dirección URL de algún recurso existente).

En el apartado anterior se ha creado un usuario de prueba, este es con el que se accede durante toda la experimentación.

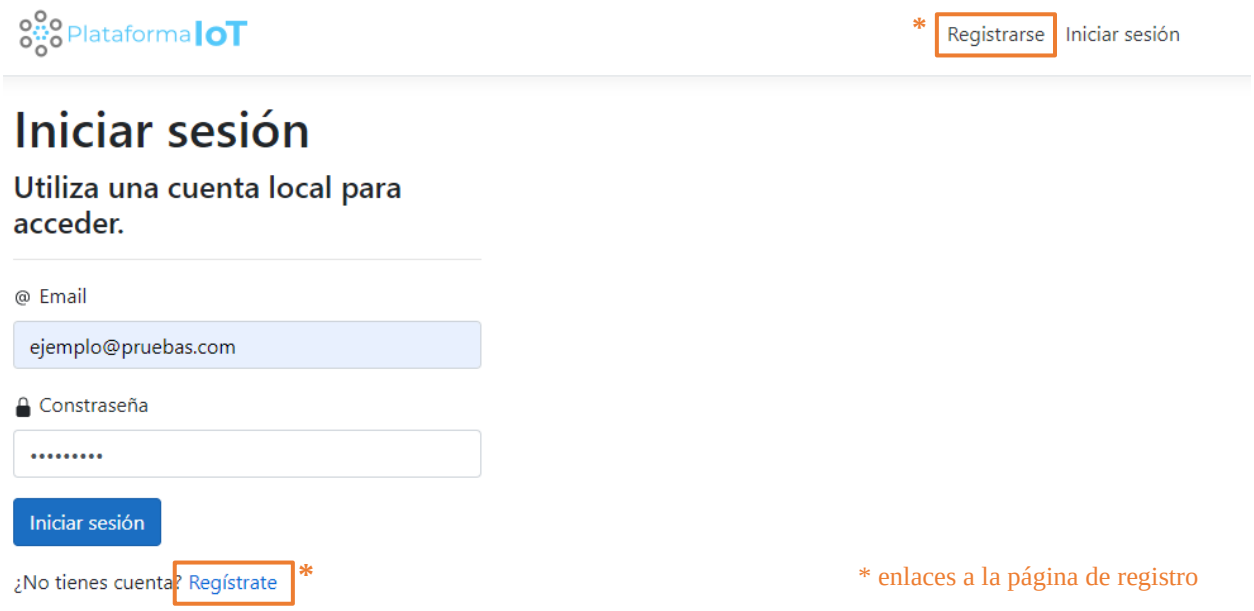


Figura 26. Acceso a la página web

Además, cada usuario del sistema solamente tiene acceso a sus recursos. Para comprobarlo, se ha intentado acceder con el usuario ejemplo@pruebas.com (sin proyectos asociados en ese momento) a información de la estación base con id 0 perteneciente al usuario prueba@mail.com y el resultado obtenido ha sido el mensaje de error que se puede ver en la Figura 27.

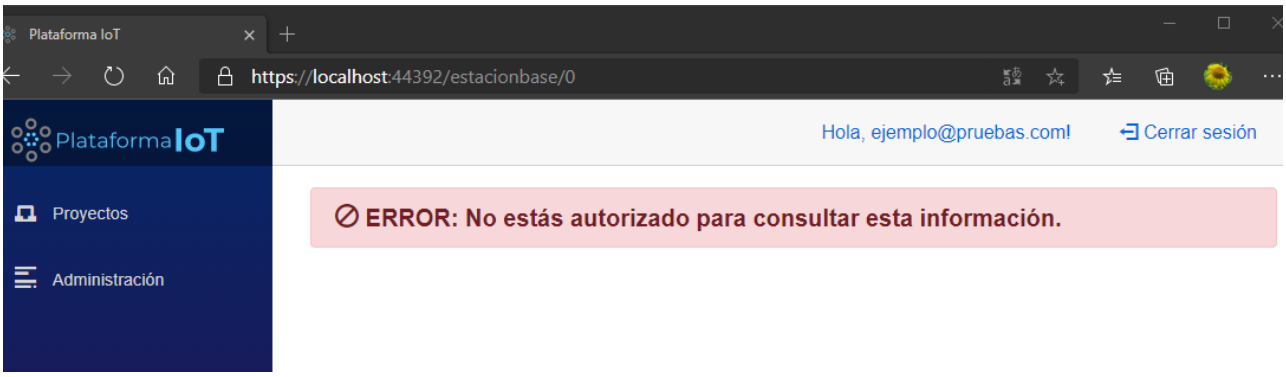


Figura 27. Intento fallido de acceso a recursos de otro usuario

6.2 Administración de proyectos

6.2.1 Crear un nuevo proyecto

Las pruebas se han realizado con un proyecto de ejemplo al que está asociada la estación base simulada comentada anteriormente (ver apartado 6.1). En la página web, se ha añadido el proyecto de prueba "Plataforma IoT-1" desde la ventana "Administración" (accesible directamente desde el menú lateral).

Como se ve en la Figura 28, en la parte superior derecha se encuentra el botón “Crear proyecto”. Al hacer clic aparece un cuadro de diálogo con un formulario para rellenar con el nombre del proyecto y una breve descripción.

Figura 28. Creación de un nuevo proyecto

Si todo va bien y el proyecto se crea correctamente aparece automáticamente entre los proyectos disponibles para editarlo (ver apartado 6.2.4) si se desea modificar algún dato o eliminarlo directamente (ver apartado 6.2.5).

En la Figura 29 se muestra el proyecto que se ha creado sin ninguna estación base asociada.

Figura 29. Proyecto de ejemplo creado

6.2.2 Añadir una estación base al proyecto

Una vez creado el proyecto, se le ha añadido una estación base desde la opción "+ añadir estación base" que aparece en cada tarjeta de proyecto.

Para agregar una nueva estación base sólo es necesario poner el nombre como se ve en la Figura 30. El nombre de las estaciones base debe tener un máximo de 10 caracteres y, según la nomenclatura establecida durante la fase de diseño, debe seguir el patrón "EBXX//EBXXX" donde las X representan el número de estación base que es.

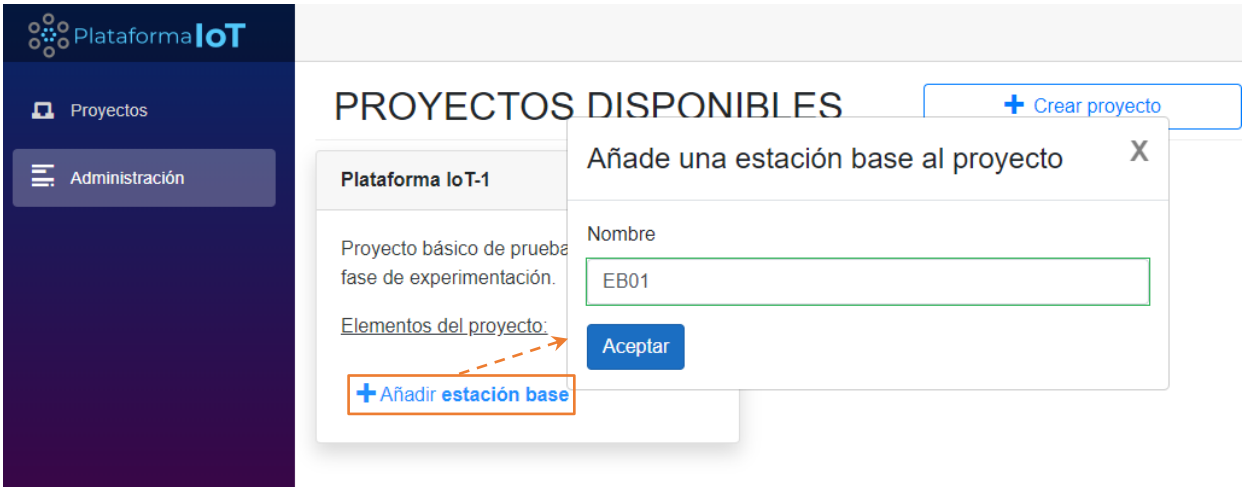


Figura 30. Adición de una nueva estación base a un proyecto existente

Si no se produce ningún error, la estación base se añade al proyecto y se muestra directamente en la tarjeta. En la Figura 31 se muestra el resultado de añadir la estación "EB01" al proyecto que se ha creado antes.

Figura 31. Estación base de ejemplo añadida al proyecto

6.2.3 Añadir un sensor a una estación base

Una vez añadida la estación base, se ha creado el sensor "SE01" haciendo uso del signo + (en color azul) que aparece junto al nombre de cada estación base. Al hacer clic se abre un cuadro de diálogo como el de la Figura 32 donde se ha ingresado el nombre del sensor y las coordenadas donde está situado.

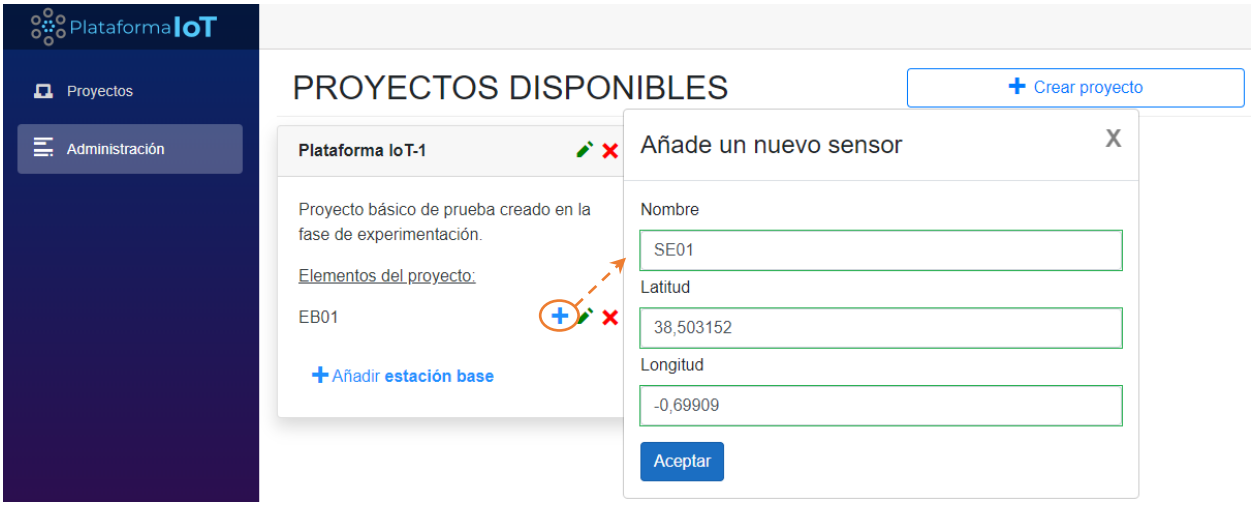


Figura 32. Adición de un nuevo sensor a una estación base

El nombre del sensor debe tener como máximo 10 caracteres y las coordenadas (como se ve en la Figura 32) se representan según latitud y longitud. La latitud se corresponde con la localización dirección Norte o Sur desde el ecuador, por lo tanto, pueden ser valores desde -90 hasta 90. Por otro lado, la longitud se corresponde con la localización dirección Este u Oeste desde el meridiano de Greenwich, pudiendo comprender los valores -180 a 180.

En la Figura 33 se puede ver el proyecto completo que se ha utilizado para mostrar las pruebas realizadas.

Figura 33. Sensor de ejemplo añadido a estación base existente

6.2.4 Editar un elemento

La página web posibilita la modificación cualquier dato de los proyectos (nombre y descripción), de las estaciones base (nombre) y de los sensores (nombre y coordenadas) a través del icono en forma de lápiz en color verde que aparece junto al nombre de cada uno de los elementos (Figura 34).

Al hacer clic sobre el lápiz aparece un *pop-up* como el de la Figura 34 que permite hacer cambios sobre los datos anteriores. Una vez que se han aceptado los cambios, estos se actualizan de manera automática en la tarjeta del proyecto.

Figura 34. Botones para editar/eliminar los elementos disponibles

6.2.5 Eliminar un elemento

Para eliminar un proyecto, una estación base o un sensor es igual en todos los casos: al lado del nombre de cada elemento aparece (junto al lápiz verde comentado en apartados anteriores) una x de color rojo (ver Figura 34. Botones para editar/eliminar los elementos disponibles, al hacer clic sobre ella elimina el elemento junto al que aparece.

Si se decide borrar un proyecto se eliminarán consecuentemente todas sus subelementos asociados (estaciones base y sensores).

6.2.6 Generar claves RSA

En la misma página de administración de proyectos, en la parte superior derecha, aparece el botón "Generar claves RSA". Al pulsar el botón, las claves se descargan automáticamente como se puede ver en la Figura 35.

Cada vez que se pulsa el botón se genera un par de claves nuevo y para utilizarlos se deben llevar a sus directorios correspondientes en la estación base (clave pública) y el servidor (clave privada).

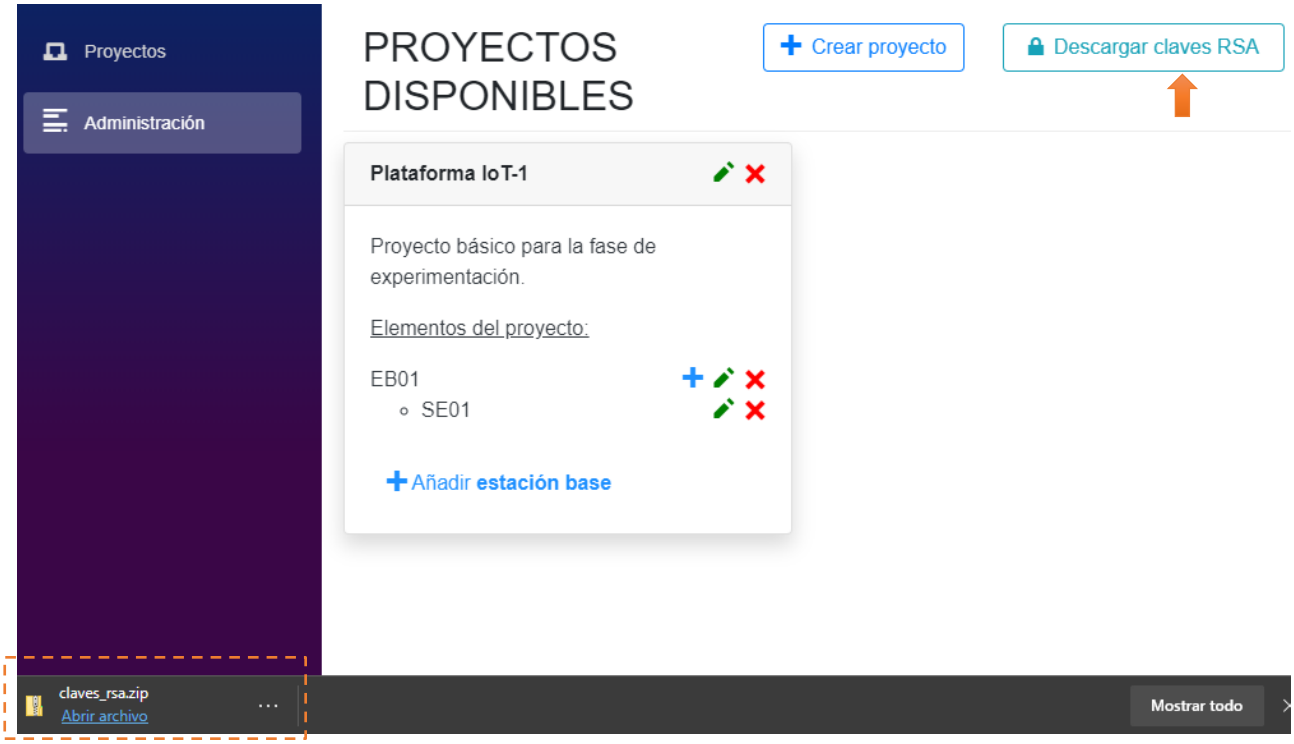


Figura 35. Descarga de claves desde la ventana de administración

6.3 Visualización de datos

La parte principal de la aplicación web es la visualización de los datos, en este apartado se muestran las vistas relacionadas con la representación de la información desde las páginas más generales a las páginas de datos más concretos.

6.3.1 Proyectos

En la Figura 36 se muestra un ejemplo de la vista principal de la aplicación. En ella se muestra una lista con todos los proyectos que posee. En este caso, solamente se ha creado el proyecto “Plataforma IoT-1”.

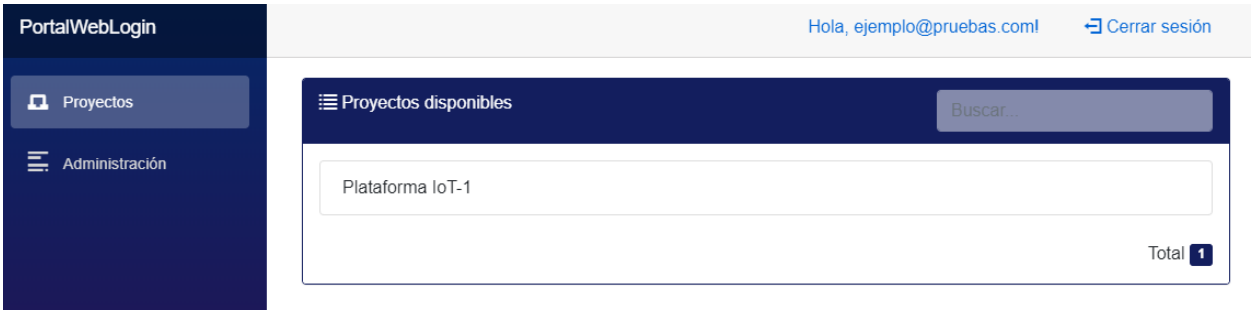


Figura 36. Lista con los proyectos creados por el usuario

Cuando se pulsa sobre algún proyecto, en la parte inferior de la vista aparecen en forma de tarjetas las estaciones base que forman parte de ese proyecto (Figura 37. Lista de estaciones base que componen el proyecto seleccionado). En cada tarjeta se muestra el nombre de la estación base, el nombre del proyecto al que pertenece (para evitar posibles confusiones) y el número total de sensores asociados a la estación; de esta forma el usuario dispone de un vistazo rápido información básica de un proyecto.

Figura 37. Lista de estaciones base que componen el proyecto seleccionado

6.3.2 Estación base

En esta vista se muestra información más detallada sobre los sensores asociados a la estación base seleccionada. En esta página, como se puede ver en la Figura 38, el elemento principal es el mapa donde se muestra la ubicación exacta de los sensores pertenecientes a la estación. En el mapa, al pasar el ratón por encima de cada marcador, aparece el nombre del sensor para facilitar su distinción.

Debajo del mapa se muestra cada uno de los sensores de la estación base en una tarjeta con información básica del último registro y un botón para acceder a su página concreta (ver apartado 6.3.3).

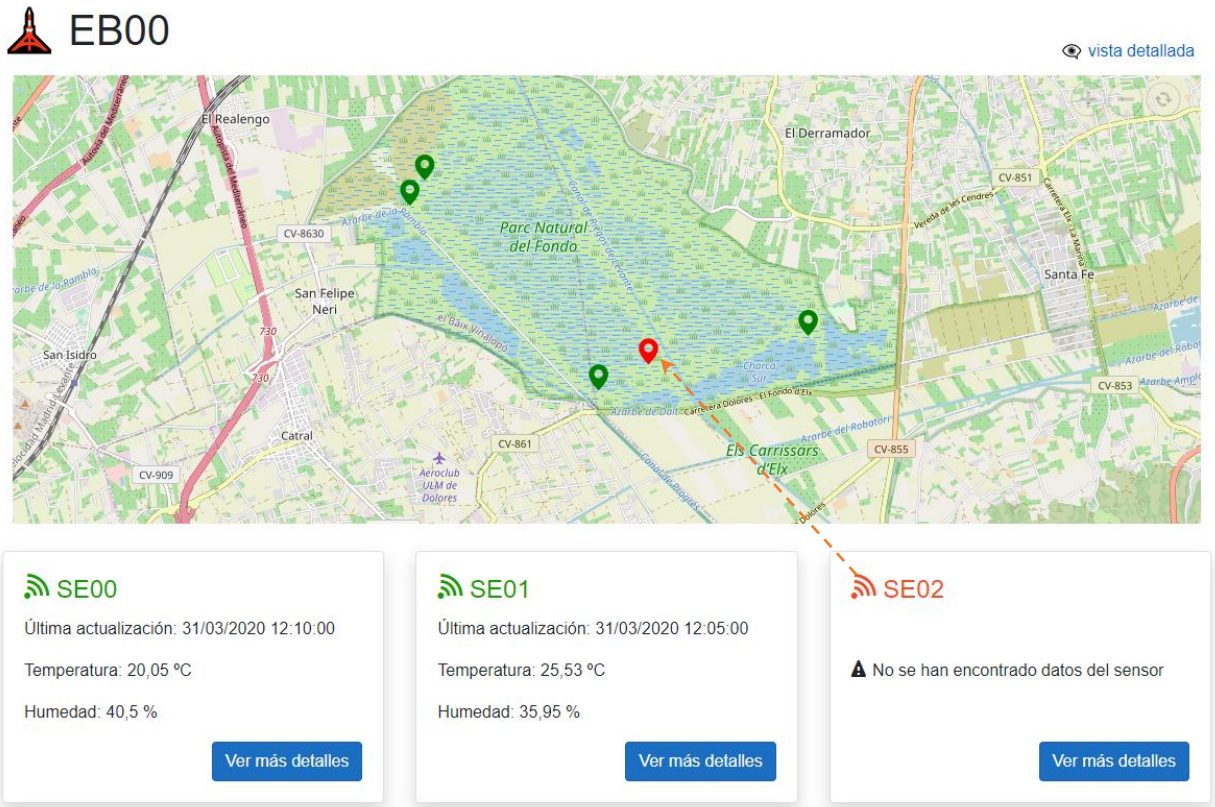


Figura 38. Vista de una estación base de ejemplo con sus sensores

En la Figura 38, el nombre de los sensores y los marcadores aparecen en verde señalan que la aplicación dispone de datos procedentes de estos sensores. El color rojo denota la falta de datos.

En el caso de que no se haya agregado ningún sensor a la estación base, el mapa aparece vacío como en la Figura 39.

Figura 39. Vista de una estación base sin sensores asociados

Por otro lado, se dispone de una vista más analítica de la estación base seleccionada haciendo clic en la opción "vista detallada" que aparece en la parte superior derecha de la página (Figura 38. Vista de una estación base de ejemplo con sus sensores). Este enlace redirige a una nueva página con dos gráficas (una para temperatura y otra para humedad) que representan una media ponderada de todos los datos tomados por los sensores de esta estación comprendidos entre dos fechas a escoger. En la Figura 40 se muestra una gráfica de temperatura de ejemplo.

Como se ve en la Figura 40, cada sensor aparece de un color en la gráfica para que se puedan diferenciar fácilmente. Además, para que la visualización de la información sea más cómoda e intuitiva en la gráfica de temperatura se utiliza el mismo color para representar los mismos sensores.

Figura 40. Gráfica comparativa de todos los datos de una estación base

En cada gráfica el usuario puede quitar la representación de los sensores que considere solamente pulsando en la gráfica el nombre de aquellos sensores de los que no quiera ver información. En la Figura 41 se han eliminado dos sensores a modo de ejemplo en la gráfica de humedad.

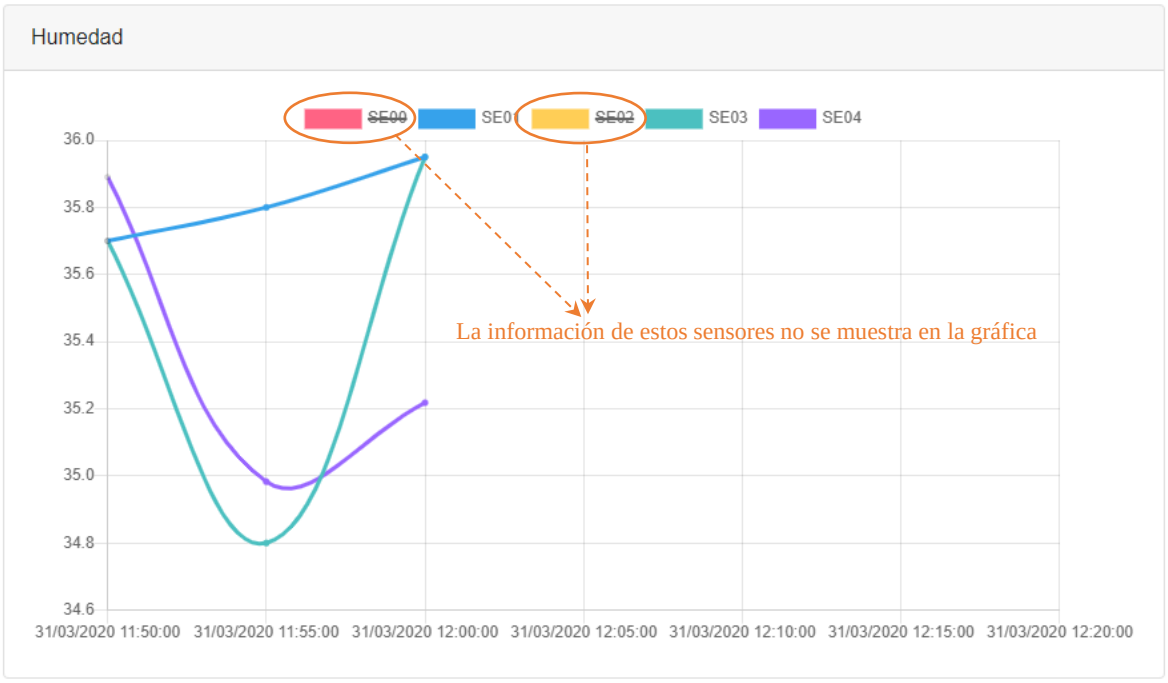


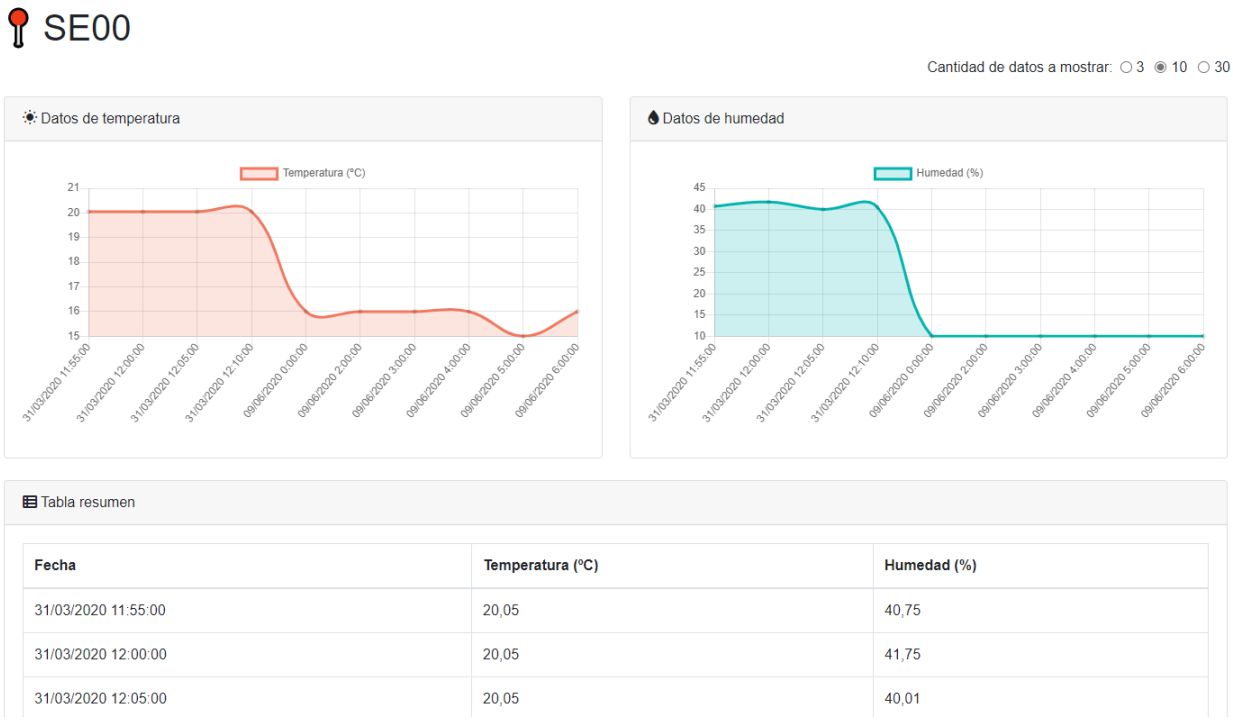
Figura 41. Gráfica comparativa de datos de sensores concretos de una estación base

6.3.3 Datos de un sensor

La información más detallada se ve en la vista de cada uno de los sensores. En esta vista (Figura 42), la información aparece representada en dos gráficas diferentes: una para los datos de temperatura y otra para los datos de humedad. Esto permite analizar los datos de forma rápida y visual.

Además, los datos también aparecen representados en una tabla para que la información se puede analizar de manera más clara.

Como se puede ver en la Figura 42, es posible ajustar la cantidad de datos que se quiere visualizar: últimos 3, 5, 10 o 20 registros del sensor.



Fecha	Temperatura (ºC)	Humedad (%)
31/03/2020 11:55:00	20,05	40,75
31/03/2020 12:00:00	20,05	41,75
31/03/2020 12:05:00	20,05	40,01

Figura 42. Vista detallada de los datos de un sensor concreto

Los datos que se muestran en la Figura 42 se corresponden con los datos almacenados en el documento de prueba comentado anteriormente.

7 Conclusiones

Para concluir el proyecto se va a realizar un pequeño análisis del trabajo realizado y a comentar diferentes líneas de ampliación y trabajos futuros para mejorar el proyecto actual.

La idea principal de este proyecto era ofrecer un pequeño portal de administración y visualización de datos de dispositivos IoT, más concretamente para el tratamiento de datos de temperatura y humedad en estaciones base.

Para alcanzar los objetivos propuestos se ha con podido desarrollar una aplicación web totalmente funcional que cubre las necesidades básicas propuestas de visualización y gestión de los datos. Para ello se han realizado las siguientes tareas:

- Diseñar un sistema de información completo a partir de unos requerimientos previos: diseño de la base de datos, escoger el modelo de comunicación entre los elementos, establecer las herramientas y lenguajes …
- Estudiar algunos de los protocolos de comunicación IoT existentes con el objetivo de entender cómo funcionamiento y su objetivo para poder escoger el que mejor se adaptara al sistema que se iba a desarrollar.
- Crear una API para recibir la información y almacenarla en la base de datos, así como otros recursos como servicios de acceso a datos (utilizados tanto por la API como por la aplicación web para el acceso a datos), interfaces, etc. para conseguir el máximo desacoplamiento posible (consiguiendo una mejor mantenibilidad del código, portabilidad a otros lenguajes, etc.).
- Desarrollar una aplicación web para que se puedan ver los datos captados por los sensores y enviados por la estación base a un servidor donde se encontrarían la base de datos y los recursos antes mencionados.

Con la solución propuesta, se han conseguido los objetivos manteniendo siempre las bases de un entorno seguro y de alto rendimiento; tanto los sistemas de seguridad como los protocolos de comunicación son ampliamente utilizados hoy día y han demostrado su fiabilidad y, además, toda la implementación se ha realizado en base a tecnologías Microsoft multiplataforma, ofreciendo un punto de vista distinto (diferentes lenguajes/herramientas) al tratado en la universidad, pero siguiendo el principio de desarrollo multiplataforma.

Una de las tecnologías con las que se ha trabajado es Blazor, una herramienta reciente e interesante para explorar. Ha facilitado mucho el desarrollo web ya que se desarrolla en el mismo lenguaje de programación en el que está desarrollado el resto del proyecto (desarrollo *full-stack* en C#). Además, al trabajar con Visual

Studio se ha conseguido acelerar de manera considerable el proceso de desarrollo ya que provee de una serie de implementaciones base muy completas de las que partir.

Con el desarrollo de este proyecto se han adquirido conocimientos muy variados además de ponerse en práctica algunos de los introducidos en la universidad, por ejemplo:

- Entender cómo funciona y cómo se despliega una API.
- Aprender a desarrollar y desplegar un servicio autónomo en un servidor Linux (Worker).
- Poner en práctica el desarrollo en capas visto previamente en la universidad y aplicar nuevos patrones de diseño (MVVM) que hasta ahora solamente habían sido tratados de manera teórica.
- Utilizar y adaptar algoritmos de cifrado robustos aprendidos en la carrera para la securización de la comunicación.
- Conocer algunos de los patrones más utilizados en el paradigma IoT.

Por último, se comentan algunos aspectos en los que puede continuar trabajando y algunas mejoras para enriquecer la experiencia de uso de la página web. Al tratarse de una página principalmente para visualizar datos, los trabajos de continuidad más evidentes son los que añaden funcionalidades a la aplicación para el tratamiento/visualización de los mismos.

Como se ha comentado anteriormente en el documento, el proyecto estaba orientado a ser desplegado en un entorno real, pero no se ha podido llevar a cabo debido a la situación de pandemia sufrida. Se dispone de una estación base real con una serie de sensores desplegados que están funcionando y captando datos. Sin embargo, al no poder acceder a los materiales del laboratorio no se han podido desplegar los servicios en este entorno y las pruebas han tenido que ser simuladas (los ficheros de datos de sensores han sido creados a mano con datos aleatorios y la estación base se ha emulado en una máquina virtual Ubuntu). Por lo tanto, la continuidad inmediata del proyecto sería trasladar la plataforma desarrollada al caso de uso real.

Actualmente, la página web hace uso del paquete *NuGet* de Syncfusion para mostrar los mapas. Este NuGet se puede utilizar gratuitamente bajo ciertas condiciones (*Community License*[17]). No obstante, esta extensión en poco estable en entornos Linux y una mejora a futuro sería encontrar una extensión alternativa más robusta o implementar una propia.

Una de las posibles ampliaciones es la compartición de proyectos entre usuarios. Durante el diseño de la base de datos se tuvo en cuenta la posibilidad de que usuarios registrados en el sistema pudieran compartir proyectos para poder acceder a la información sin necesidad de crear un proyecto idéntico cada usuario; sin

[17] https://www.syncfusion.com/products/communitylicense

embargo, esta funcionalidad no está implementada en proyecto dado que no formaba parte del diseño inicial. Otra posible ampliación del proyecto relacionada con esta misma idea es contemplar la opción de añadir roles a los usuarios para que sólo los usuarios que con categoría superior puedan eliminar/modificar un proyecto.

En lo referente a la funcionalidad ofrecida al usuario, una de las ideas iniciales planteadas que no se ha podido implementar ha sido enviar alertas al usuario en el caso de que se detecte algún fallo o interrupción en la recepción de los datos en algún sensor o estación base. Sin embargo, el usuario es capaz de ver que se han producido cortes en la comunicación revisando la información en las gráficas.

Otras líneas de mejoras para ampliar la experiencia del usuario serían: extender las posibilidades de filtrado de la información de la página web, añadir la posibilidad de descargar los documentos y otras opciones de visualización (gráficos de barras, de área, etc.).

8 Apéndice

En esta sección se explican algunos conceptos y términos que aparecen en el informe.

NuGet: gestor de paquetes que facilita la instalación y administración de librerías .NET. Estas librerías pueden ser de código abierto (por ejemplo, Blazorise) o privadas (por ejemplo, Syncfusion).

IoT (*Internet of Things*): término que hace referencia a la interconexión de dispositivos (sensores, electrodomésticos, máquinas, etc.) a través de la red.

OASIS (*Organization for the Advancement of Structured Information Standards*): organización sin ánimo de lucro para el desarrollo y la adopción de estándares mundiales abiertos para la seguridad, el internet de las cosas, la computación en la nube …

IETF (*Internet Engineering Task Force*): organismo encargado de definir protocolos estándar de Internet como TCP/IP.

WebAssembly: formato de código binario de bajo nivel, independiente del lenguaje (compilado desde C/C++ y Rust) diseñado para ser ejecutado en la web aportando un rendimiento casi nativo (superior al ofrecido por JavaScript, un lenguaje interpretado).

SignalR: librería de código abierto que añade funcionalidad web en tiempo real a las aplicaciones desarrolladas en ASP.NET Core. Esto es, facilita el envío de código del lado del servidor al cliente de manera inmediata cuando esté disponible.

ORM (*Object-Relational Mapping*): modelo de programación especializado en el acceso a datos encargado de la conversión de los objetos a un formato apropiado que pueda ser almacenado en una base de datos relacional para simplificar la tarea de acceso a datos al programador.

9 Bibliografía

[1] MQTT. https://mqtt.org/

[2] Michael Yuan. (2017). Conozca MQTT de IBM Developer. https://developer.ibm.com/es/technologies/iot/articles/iot-mqtt-why-go

[3] CoAP. http://coap.technology/

[4] MarkelIglesias-Urkia, Adrián Orive, AitorUrbieta (2017). *Analysis of CoAP Implementations for Industrial Internet of Things: A Survey*. https://www.sciencedirect.com/science/article/pii/S1877050917309870

[5] Margaret Rouse. *Constrained Application Protocol* (CoAP) de WhatIs. https://whatis.techtarget.com/definition/Constrained-Application-Protocol

[6] Internet Engineering Task Force (IETF), *The Constraint Application Protocol (CoAP)*. RFC 7252. https://tools.ietf.org/html/rfc7252

[7] Internet Engineering Task Force (IETF), *The Constraint Application Protocol (CoAP)*. RFC 7252, Sección 5.9. https://tools.ietf.org/html/rfc7252#section-5.9

[8] Jonathan Fries. *Why are IoT developers confused by MQTT and CoAP?*. 04 May 2017, de IoT Agenda. https://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/Why-are-IoT-developers-confused-by-MQTT-and-CoAP

[9] Microsoft. *The Model-View-ViewModel Pattern*. https://docs.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[10] Wikipedia. Modelo-Vista-Controlador. https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93modelo_de_vist

[11] Ecured. Modelo Vista Vista Modelo. https://www.ecured.cu/Modelo_Vista_Vista_Modelo

[12] Microsoft. *What is .NET Framework?*. https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework#:~:text=%20.NET%2C%20.NET%20Framework%2C%20and%20.NET%20Core%20,on%20all%20the%20major%20mobile%20operating...%20More%20

[13] José María Aguilar (2019). 10 diferencias entre .NET Core y .NET Framework de Campus MVP. https://www.campusmvp.es/recursos/post/10-diferencias-entre-net-core-y-net-framework.aspx

[14] Microsoft. *.NET Core overview*. https://docs.microsoft.com/es-es/dotnet/core/about

[15] NuGet CoAP.NET.Core. https://www.nuget.org/packages/CoAP.NET.Core

[16] SmeshLink (Github). *CoAP.NET documentation*. http://open.smeshlink.com/CoAP.NET/

[17] Microsoft. *AES Class*. https://docs.microsoft.com/es-es/dotnet/api/system.security.cryptography.aes?view=netcore-3.1

[18] Microsoft. *RSA Class*. https://docs.microsoft.com/es-es/dotnet/api/system.security.cryptography.rsa?view=netcore-3.1

[19] Serilog. https://serilog.net/

[20] NuGet Serilog.AspNetCore. https://www.nuget.org/packages/Serilog.AspNetCore/3.4.0-dev-00173

[21] Mukesh Murugan (2020). Serilog in ASP.NET Core 3.1 – Structured Logging Made Easy. https://www.codewithmukesh.com/blog/serilog-in-aspnet-core-3-1/

[22] Dapper ORM. https://dapper-tutorial.net/

[23] Microsoft. Blazor. https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor

[24] José María Aguilar (2020). Introducción a Blazor a través de 7 preguntas (y sus respuestas) de Campus MVP. https://www.campusmvp.es/recursos/post/Introduccion-a-blazor-a-traves-de-7-preguntas.aspx

[25] Microsoft. *Introduction to ASP.NET Core Blazor*. https://docs.microsoft.com/es-es/aspnet/core/blazor/?view=aspnetcore-3.1

[26] Daniel Roth. *ASP.NET Core Blazor hosting models* de Microsoft. https://docs.microsoft.com/es-es/aspnet/core/blazor/hosting-models?view=aspnetcore-3.1

[27] Syncfusion. https://www.syncfusion.com/

[28] *Interactive Blazor Maps Component* de Syncfusion. https://www.syncfusion.com/blazor-components/blazor-maps

[29] Mladen Macanovic. Blazorise documentation. https://blazorise.com/docs/

[30] *Chart extension* de Blazorise. https://blazorise.com/docs/extensions/chart/

[31] Andy Belfield (2018). *Implementing RSA in .NET Core* de 4142.io. https://4142.io/Code/DotNet/RSA-in-Net-Core/

A. ANEXO I: Despliegue del Worker

En este anexo del documento se explican los pasos necesarios para preparar la estación base y poder enviar la información al servidor. En este caso, se detalla cómo realizar el despliegue en una máquina virtual Ubuntu ya que el worker no se ha podido llevar a la estación base real (una RasperryPi con Linux); en la estación base real se deberían reproducir los mismos comandos y pasos comentados en este anexo.

A.1 Dependencias .NET Core

.NET Core está soportado en sistemas Linux, para poder desarrollar o ejecutar un proyecto .NET Core simplemente es necesario descargar los paquetes para .NET (el *sdk* o el *runtime*) disponibles a través del gestor de paquetes. La versión de .NET Core utilizada para el desarrollo es la 3.1 por tanto, también se debe instalar la misma versión en Linux[18].

Antes de instalar los paquetes, es necesario añadir el repositorio de Microsoft:

```
# wget https://packages.microsoft.com/config/ubuntu/18.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb
# sudo dpkg -i packages-microsoft-prod.deb
```

Tras esto, se instala el sdk (permite desarrollar y ejecutar proyectos .NET) o el runtime (permite únicamente ejecutar aplicaciones/proyectos .NET):

```
# sudo apt-get install -y apt-transport-https
# sudo apt-get install -y dotnet-sdk-3.1
# sudo apt-get install -y aspnetcore-runtime-3.1
```

A.2 Configuración

Los binarios del servicio están comprimidos en un archivo zip que deben ser extraídos en una ruta concreta del sistema donde vaya a ser ejecutado el servicio. Además, es necesario definir una serie de directorios que se utilizarán para configurar dicho servicio (Figura 43. Directorios ejemplo del worker).

Figura 43. Directorios ejemplo del worker

[18] https://docs.microsoft.com/en-us/dotnet/core/install/linux-ubuntu

Una vez instalados los paquetes necesarios, se debe crear un archivo para definir y registrar el servicio. Dicho archivo debe estar alojado en el directorio "/etc/systemd/system" con el nombre que se quiera dar al servicio.

Para ilustrar cómo debe quedar el fichero en la Figura 44 se muestra el que se ha utilizado en la fase de experimentación.

```
  GNU nano 2.9.3                    /etc/systemd/system/worker.service

[Unit]
Description=Servicio para enviar los datos de los sensores

[Service]
WorkingDirectory=/home/ubuntu/Escritorio/Experimentacion/worker
ExecStart=/usr/bin/dotnet /home/ubuntu/Escritorio/Experimentacion/worker/EstacionBase.WorkerService.dll
Restart=always
RestartSec=10
SyslogIdentifier=dotnet-worker
User=ubuntu
Environment=ASPNETCORE_ENVIRONMENT=Production

[Install]
WantedBy=multi-user.target
```

Figura 44. Fichero de definición del servicio worker en Linux

Algunos de los atributos de configuración a tener en cuenta en el archivo son los siguientes:

- Description: breve descripción de lo que hace el servicio,
- WorkingDirectory; ruta donde se han guardado los binarios,
- ExecStart: ruta completa al binario (.dll) principal del servicio (punto de entrada),
- User: el usuario que ejecutará el servicio,
- RestartSec: tiempo establecido para el reiniciar el servicio ante un fallo del sistema

Una vez definido el servicio, es necesario configurarlo antes de poner arrancarlo. El fichero de configuración ("appsettings.json)"se encuentra junto con el resto de binarios del worker como se puede apreciar en la Figura 45.

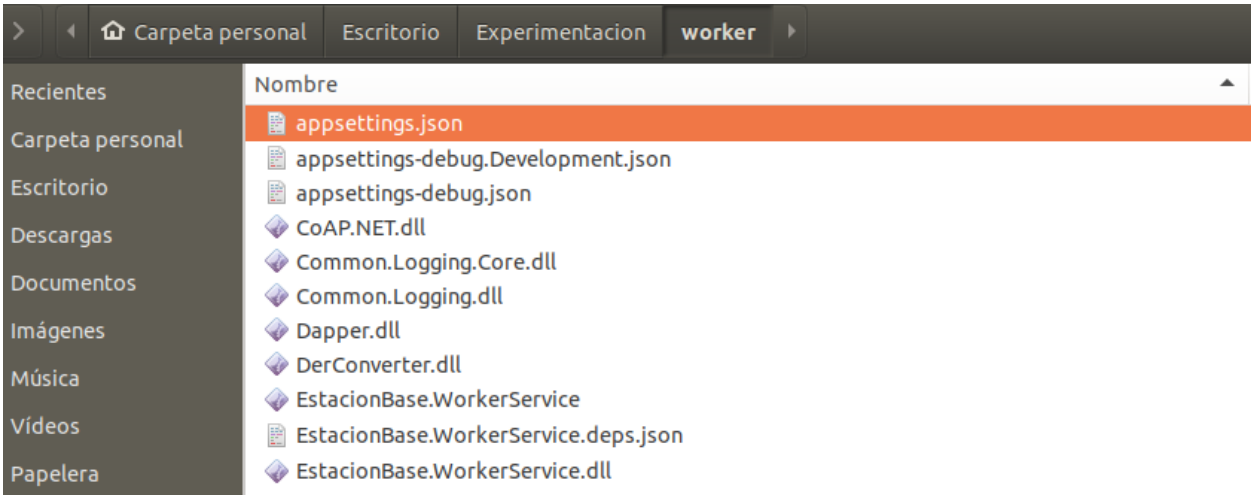


Figura 45. Directorio con el fichero de configuración del worker

En el archivo "appsettings.json" se establecen los valores de configuración de la estación base (el directorio donde se encuentra la clave, dónde almacenar el *log*, etc.). En la Figura 46 se muestra la configuración correspondiente para las pruebas realizadas donde se ve cada uno de los atributos que se debe configurar precedido de un comentario aclaratorio.

```
  GNU nano 2.9.3        /home/ubuntu/Escritorio/Experimentacion/worker/appsettings.json

{
  //Ruta del fichero de log
  "DirectorioLog": "/home/ubuntu/Escritorio/Experimentacion/worker.log",

  //URL del servidor
  "UriCoap": "coap://192.168.56.102:5683/COAPServer",

  //Tiempo que tarde el worker en leer el archivo y enviar la peticion (EN SEGUNDOS)
  "TiempoEnvio": "300",

  //Fichero donde se almacenan la clave publica
  "FicheroClaveRSA": "/home/ubuntu/Escritorio/Experimentacion/clave_rsa/publica.key",

  //Ruta de los ficheros de los sensores
  "DirectorioSensores": "/home/ubuntu/Escritorio/Experimentacion/sensores/",

  // Nombre de la estaci◆n base
  "EstacionBase": "EB01",

  //Nombre del proyeto al que pertenece la estaci◆n base
  "Proyecto": "Plataforma IoT-2"

}
```

Figura 46. Fichero de configuración del worker

El último paso de la configuración del worker es copiar la clave pública generada desde la página web. La clave debe estar contenida en un directorio propio que debe ser el mismo que el especificado en el fichero de configuración (Figura 47. Directorio en la estación base con la clave pública del servidor).

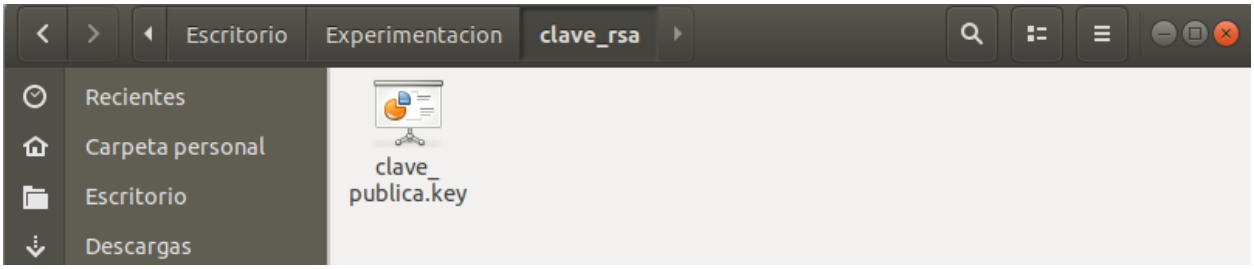


Figura 47. Directorio en la estación base con la clave pública del servidor

Para que el servicio funcione correctamente, el proyecto y la estación base deben haber sido dados de alta previamente a través de la página web y, evidentemente, la API debe estar en ejecución a la espera de peticiones.

A.3 Inicio del servicio

Cuando el demonio ya está definido y la aplicación configurada, el último paso es habilitarlo e iniciarlo para que comience a enviar información. Los comandos para habilitar e iniciar el servicio son los siguientes, en orden:

```
# sudo systemctl enable worker.service
# sudo service worker start
```

En la Figura 48 se puede ver un ejemplo de cómo se arranca el servicio y se comprueba su estado.

```
ubuntu@ubuntu:~$ sudo systemctl enable worker.service
Created symlink /etc/systemd/system/multi-user.target.wants/worker.service → /etc/systemd/system/worker.service.
ubuntu@ubuntu:~$
ubuntu@ubuntu:~$ sudo service worker start
ubuntu@ubuntu:~$
ubuntu@ubuntu:~$ sudo systemctl | grep worker
worker.service                                                                loaded active running
Servicio para enviar los datos de los sensores
ubuntu@ubuntu:~$
```

Figura 48. Ejemplo de inicio del worker

B. ANEXO II: Despliegue de la API

En este anexo se muestran los pasos a seguir para desplegar la API en un servidor Linux. Los pasos son muy similares a los que se deben seguir para desplegar el Worker.

B.1 Dependencias .NET Core y SQL Server

Se deben instalar los paquetes de .NET Core para Linux (comentados en "A. ANEXO I: Despliegue del Worker").

Además, para este anexo se ha preparado la base de datos en la misma máquina que la API, aunque podría estar en otro servidor distinto (solo habría que cambiar la cadena de conexión que se ve más adelante en el archivo de configuración).

Antes de poder descargar e instalar SQL Server es necesario agregar e importar las llaves GPU del repositorio público de Microsoft y registrarlas en el repositorio de Ubuntu:

```
# wget -qO- https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
```

```
# sudo add-apt-repository "$(wget -qO- https://packages.microsoft.com/config/ubuntu/16.04/mssql-server-2017.list)"
```

El paquete se puede instalar directamente a través de terminal con el siguiente comando:

```
# sudo apt-get update
```

```
# sudo apt-get install mssql-server -y
```

Una vez instalado, el último paso es configurar SQL con el comando que se ve a continuación:

```
# sudo /opt/mssql/bin/mssql-conf setup
```

Como se puede ver en la Figura 49 (y según lo comentado en la definición de la arquitectura de la plataforma) se ha escogido la versión SQL Server Express. Tras definir la versión, se establece la contraseña del usuario administrador de SQL Server (denominado "sa"); este usuario se puede utilizar para definir posteriormente otros usuarios de SQL Server sin privilegios de administrador.

```
ubuntu@ubuntu:~$
ubuntu@ubuntu:~$ sudo /opt/mssql/bin/mssql-conf setup
Elija una edición de SQL Server:
  1) Evaluation (gratis, sin derechos de uso en producción, límite de 180 días)
  2) Developer (gratis, sin derechos de uso en producción)
  3) Express (gratis)
  4) Web (DE PAGO)
  5) Standard (DE PAGO)
  6) Enterprise (DE PAGO)
  7) Enterprise Core (DE PAGO)
  8) He comprado una licencia mediante un canal de ventas al por menor y tengo una clave de producto.

Encontrará información sobre las ediciones en
https://go.microsoft.com/fwlink/?LinkId=852748&clcid=0x40a

Para usar las ediciones DE PAGO de este software es necesaria una licencia distinta mediante un
Programa Licencias por Volumen de Microsoft.
Al elegir la edición DE PAGO, confirma que dispone del
número adecuado de licencias para instalar y ejecutar este software.

Especifique su edición(1-8): 3
```

Figura 49. Configuración de SQL Server en Linux

Tras configurarlo, SQL Server arranca automáticamente. En la Figura 50 se muestra el estado del servicio SQL activo.

```
La configuración se ha completado correctamente. SQL Server se está iniciando.
ubuntu@ubuntu:~$ systemctl status mssql-server
● mssql-server.service - Microsoft SQL Server Database Engine
   Loaded: loaded (/lib/systemd/system/mssql-server.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2020-07-26 18:45:00 CEST; 2min 44s ago
     Docs: https://docs.microsoft.com/en-us/sql/linux
 Main PID: 5756 (sqlservr)
    Tasks: 110
   CGroup: /system.slice/mssql-server.service
           ├─5756 /opt/mssql/bin/sqlservr
           └─5795 /opt/mssql/bin/sqlservr

jul 26 18:45:43 ubuntu sqlservr[5756]: [72B blob data]
jul 26 18:45:43 ubuntu sqlservr[5756]: [66B blob data]
jul 26 18:45:44 ubuntu sqlservr[5756]: [108B blob data]
jul 26 18:45:44 ubuntu sqlservr[5756]: [96B blob data]
jul 26 18:45:44 ubuntu sqlservr[5756]: [112B blob data]
jul 26 18:45:44 ubuntu sqlservr[5756]: [100B blob data]
jul 26 18:45:45 ubuntu sqlservr[5756]: [86B blob data]
jul 26 18:45:45 ubuntu sqlservr[5756]: [71B blob data]
jul 26 18:45:45 ubuntu sqlservr[5756]: [159B blob data]
jul 26 18:45:45 ubuntu sqlservr[5756]: [124B blob data]
ubuntu@ubuntu:~$
```

Figura 50. Comprobación del estado de SQL

Además, en la solución se ha añadido un script SQL encargado de generar el esquema de la base de datos necesario para que se pueda registrar toda la información correctamente.

B.2 Configuración

La API se ha planteado como un servicio, de manera que se debe definir en un archivo dentro del directorio /etc/systemd/system. A continuación, en la Figura 51, se muestra un ejemplo de fichero de definición.

```
 GNU nano 2.9.3                    /etc/systemd/system/api.service

[Unit]
Description=API-inserta los datos en la base de datos

[Service]
WorkingDirectory=/home/ubuntu/Escritorio/Experimentacion/api
ExecStart=/usr/bin/dotnet /home/ubuntu/Escritorio/Experimentacion/api/API.dll
Restart=always
RestartSec=10
SyslogIdentifier=dotnet-api
User=ubuntu
Environment=ASPNETCORE_ENVIRONMENT=Production

[Install]
WantedBy=multi-user.target
```

Figura 51. Definición del servicio de la API

Una vez definida, se debe configurar en el archivo "appsettings.json" (ver Figura 52. Fichero de configuración de ejemplo de la API) que, al igual que en el Worker, se encuentra en el mismo directorio que los binarios de la aplicación.

```
 GNU nano 2.9.3                  /home/ubuntu/Escritorio/Experimentacion/api/appsettings.json

{
  "AllowedHosts": "*",

  //Fichero donde se almacena el log [directorio_actual/api.log]
  "DirectorioLog": "/home/ubuntu/Escritorio/Experimentacion/api.log",

  //Puerto en el que se inicia el servidor CoAP
  "Puerto": 5683,

  //Cadena de conexion con la BD
  "CadenaConexion": "Data Source=localhost;Initial Catalog=plataforma_iot;Integrated Security=false; User Id=sa; Password=123456Aa.",

  //Fichero donde se almacena la clave privada
  "FicheroClaveRSA": "/home/ubuntu/Escritorio/Experimentacion/clave_rsa/privada.key"
}
```

Figura 52. Fichero de configuración de ejemplo de la API

En el caso de la API, para que pueda descifrar la información recibida es necesario crear un nuevo directorio para almacenar la clave privada como se muestra en la Figura 53.

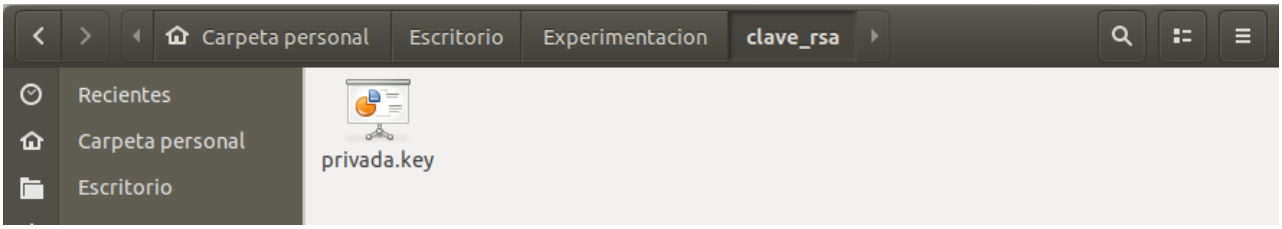


Figura 53. Directorio con la clave privada en el servidor

B.3 Arranque

Por último, cuando se ha configurado el servicio de la API, se puede habilitar e iniciar para que comience a esperar las peticiones procedentes de las estaciones base (workers). Los comandos necesarios son los siguientes:

```
# sudo service api enable
# sudo service api start
```

En la Figura 54 se muestra cómo la API configurada está a la escucha de peticiones.

```
ubuntu@ubuntu:~$ sudo service api status
● api.service - API-inserta los datos en la base de datos
   Loaded: loaded (/etc/systemd/system/api.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2020-07-27 12:43:19 CEST; 3s ago
 Main PID: 3349 (dotnet)
    Tasks: 12 (limit: 3535)
   CGroup: /system.slice/api.service
           └─3349 /usr/bin/dotnet /home/ubuntu/Escritorio/Experimentacion/api/Servidor.API.dll

jul 27 12:43:19 ubuntu systemd[1]: Started API-inserta los datos en la base de datos.
ubuntu@ubuntu:~$
```

Figura 54. Comprobación del estado de la API

C. ANEXO III: Despliegue de la página web

En este anexo se comenta cómo realizar el despliegue de la aplicación web desarrollada en Blazor en un servidor web Linux (Ubuntu, en este caso).

C.1 Dependencias

En primer lugar, es necesario instalar las dependencias .NET Core para que se pueda ejecutar. En el "A. ANEXO I: Despliegue del Worker" se explica cómo instalar paso a paso los paquetes necesarios.

Para establecer un servidor web en Ubuntu que hospede la página se debe buscar servidores web compatibles con Blazor: apache, nginx o IIS (*Internet Information Server)*.

En este caso, se ha configurado un servidor web apache ya que viene instalado en el sistema operativo.

C.2 Configuración del servidor web

Para poder configurarlo con el proxy inverso para para la aplicación web .NET Core, se debe habilitar los siguientes módulos:

```
# sudo a2enmod proxy proxy_http proxy_html proxy_wstunnel
```

Una vez habilitados, se debe crear la configuración del servidor en el directorio /etc/apache2/conf-available y para habilitarlo se ejecuta el siguiente comando:

```
# sudo a2enconf /etc/apache2/conf-available/servidor_web.conf
```

Para conseguir que el servidor web administre la aplicación web, se debe crear como un servicio en el directorio de configuración /etc/systemd/system. En la Figura 55 se muestra un fichero de configuración de ejemplo.

```
  GNU nano 2.9.3                    /etc/systemd/system/servidor_web.service

[Unit]
Description= ASP.NET Core Web

[Service]
WorkingDirectory=/home/ubuntu/Escritorio/Experimentacion-web/web
ExecStart=/usr/bin/dotnet /home/ubuntu/Escritorio/Experimentacion-web/web/PortalWebLogin.dll
Restart=always
RestartSec=10
SyslogIdentifier=dotnet-web
User=ubuntu
Environment=ASPNETCORE_ENVIRONMENT=Production

[Install]
WantedBy=multi-user.target
```

Figura 55. Configuración del servicio web en Ubuntu

Una vez definido y configurado, se inicia y se comprueba que se está ejecutando:

```
# sudo service servidor_web.service start
# sudo service servidor_web.service status
```

En la Figura 56 se muestra el servicio activo y se puede ver las URL desde las que se puede acceder a la página web (https://localhost:5001 o http://localhost:5000).

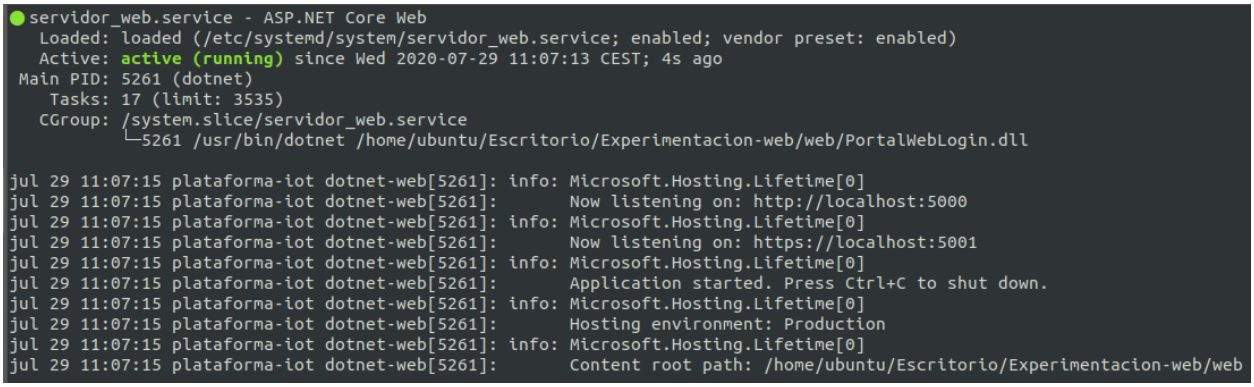


Figura 56. Estado del servicio definido para la página web